# Trusted Computing Platform Alliance

*(TCPA)*

## *Main Specification Version 1.1a*

# Acknowledgement

TCPA wishes to thank members of the PKI, PC Specific and Conformance Workgroup who contributed expertise and text to this document. Thanks must be given to the members of the TCPA Technical Committee who were Michael Angelo, Boris Balacheff, Josh Benaloh, David Challener, Dhruv Desai, Paul England, David Grawrock, Bob Meinschein, Manny Novoa, Graeme Proudler, Jim Ward and Monty Wiseman.

David Chan

Technical Committee Chair

# Change History

| Version | Date | Description |
|---|---|---|
| 0.44 | July 2000 | Voted by members as appropriate for public release with modifications. |
| 0.90 | August 2000 | First version released to public. |
| 0.91 | October 26, 2000 | Remove chapters 1 & 2. Complete reformat |
| 0.92 | 4 November, 2000 | Added new chapter for structures, updated functions to match IDL, editing changes. |
| 1.0 RC1 | 28 November 2000 | Incorporated comments cleaned up structures and made ready for publication. |
| 1.0 RC2 | 11 December 2000 | Incorporated changes from reflector. Added new change authorization command. |
| 1.0 RC4 | 10 Jan 2001 | Incorporated changes and fixed up IDL |
| 1.0 RC5 | 11 Jan 2001 | PKCS#1 changes |
| 1.01 | 17 April 2001 | Implemented corrections. Mid point save made to avoid problems with track changes in document |
| 1.02 | 18 April 2001 | Continue with changes for 1.1 release, changed IDL to table format |
| 1.03 | | First attempt to reconcile IDL misses |
| 1.04 | 7 May 2001 | Mid level drop to show all changes in regard to IDL |
| 1.06 | 17 May 2001 | All CR's complete |
| 1.07 | 22 May 2001 | Cleanup from WG messages and changing in Audit commands. |
| 1.1 RC1 | 25 May 2001 | Release candidate for specification |
| 1.1 RC2 | 4 June 2001 | All changes |
| 1.1 RC3 | 12 June 2001 | Removal of TSS commands, cleanup of parameter blocks, all comments from v1 reflector. |
| 1.1 RC4 | 5 July 2001 | Editing changes, candidate for final review |
| 1.1 RC6 | 17 July 2001 | All changes made and version ready for voting |
| 1.1 | 31 July 2001 | Voted on release of 1.1 |
| 1.1a | 12th November 2001 | Includes all errata upto and including #55 |

# Table Of Contents

# 1. Forward

This document is an industry specification that enables trust in computing platforms in general.

This specification defines a trusted *Subsystem* that is an integral part of each platform, and provides functions that can be used by enhanced operating systems and applications. The Subsystem employs cryptographic methods when establishing trust, and while this does not in itself convert a platform into a secure computing environment, it is a significant step in that direction.

Standardization is necessary so that the security and cryptographic community can assess the mechanisms involved, and so that customers can understand and trust the effectiveness of new features. Manufacturers will compete in the marketplace by installing Subsystems with varying capabilities and cost points. The Subsystem itself will have basic functions that maintain privacy, yet support the identity and authentication of entities such as the platform, the user, and other entities. The Subsystem will have other capabilities to protect data and verify certain operational aspects of the platform. It can be a separate device or devices, or it can be integrated into some existing component or components provided the implementation meets the requirements of this specification. This is necessary to achieve the fundamental goal of ubiquity.

Please note a very important distinction between different sections of text throughout this document. Beginning in chapter 2, "The Trusted Platform Subsystem," you will encounter two distinctive kinds of text: *informative comment* and *normative statements*. Because most of the text in this specification will be of the kind *normative statements*, the authors have informally defined it as the default and, as such, have specifically called out text of the kind *informative comment*. They have done this by flagging the beginning and end of each *informative comment* and highlighting its text in gray. This means that unless text is specifically marked as of the kind *informative comment*, you can consider it of the kind *normative statements*.

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in the chapters 2-10 normative statements are to be interpreted as described in [RFC-2119].

For example:

This is the first paragraph of 1–n paragraphs containing text of the kind *informative comment* ...

This is the second paragraph of text of the kind *informative comment* ...

This is the nth paragraph of text of the kind *informative comment* ...

To understand the TCPA specification the user must read the specification. (This use of MUST does not require any action).

This is the first paragraph of one or more paragraphs (and/or sections) containing the text of the kind *normative statements* ...

To understand the TCPA specification the user MUST read the specification. (This use of MUST indicates a keyword usage and requires an action).

# 2. The Trusted Platform Subsystem

## 2.1 Introduction

*Start of informative comment:*

The TCPA Subsystem design is to provide useful trust and security capabilities while minimizing the number of functions that must be trusted. This arrangement is necessary to make the Subsystem useful while remaining low in cost and can result in unusual features as compared with a conventional crypto co-processor.

*End of informative comment.*

## 2.2 Roots of Trust

*Start of informative comment:*

This section introduces the architectural aspects of a Trusted Platform that enable the collection and reporting of integrity metrics.

Among other things, a Trusted Platform enables an entity to determine the state of the software environment in that platform and to SEAL data to a particular software environment in that platform.

The entity deduces whether the state of the computing environment in that platform is acceptable and performs some transaction with that platform. If that transaction involves sensitive data that must be stored on the platform, the entity can ensure that that data is held in a confidential format unless the state of the computing environment in that platform is acceptable to the entity.

To enable this, a Trusted Platform provides information to enable the entity to deduce the software environment in a Trusted Platform. That information is reliably measured and reported to the entity. At the same time, a Trusted Platform provides a means to encrypt cryptographic keys and to state the software environment that must be in place before the keys can be decrypted.

Both these functions require integrity metrics. These metrics consist of data reflecting the integrity of the software state of the Trusted Platform. Both functions require two roots of trust in a platform. One is known as the 'root of trust for measuring integrity metrics', and the other is known as the 'root of trust for storing and reporting integrity metrics'.

The root of trust for measuring integrity metrics is likely to be different for different types of platforms because the metrics and their measurements will depend on the type of platform. The root of trust for storing and reporting integrity metrics enables integrity metrics to be reliably stored and reported and can have the same capabilities irrespective of the type of platform.

A trusted measurement root measures certain platform characteristics, logs the measurement data in a measurement store, and stores the final result in a TPM (which contains the root of trust for storing and reporting integrity metrics). The trusted measurement root might also measure the characteristics of another measurement agent before passing control to the second agent. That second agent might repeat the process of measuring platform characteristics, storing measurement data and the final result, passing control to a third measurement agent, and so on.

When an integrity challenge is received, the Trusted Platform Agent gathers the following:

- the final results from the TPM,
- the log of the measurement data from the Trusted Platform Measurement Store, and
- TCPA Validation Data that states the values that the measurements should produce in a platform that is working correctly.

The Trusted Platform Agent then sends this measurement data to the Challenger. The Challenger uses the data to check that it is consistent with the final results and then compares the data (and perhaps the final results) with the TCPA Validation Data. This comparison enables the Challenger to deduce the

software state of the Trusted Platform and consequently decide whether the Challenger is satisfied to trust the platform for the intended purpose.

Once the Challenger has determined that the Trusted Platform can be trusted, the Challenger can use the TPM to store keys alongside stated values of integrity metrics, such that the TPM will not release the keys unless the current measured values of integrity metric match the stated values of integrity metric.

Both roots of trust, plus certain other capabilities for other purposes, must be implemented in ways that enable confidence in their correct operation in all circumstances of interest. A Challenger must be able to trust the roots and these capabilities. The implementation of the root of trust for measurement will typically vary depending on the type of platform (for example, PC, server, or phone). The TPM is defined as the set of all trusted capabilities apart from the root of trust for measurement, because these are independent of the type of platform. The whole Subsystem, therefore, typically consists of a root of trust for measuring integrity metrics, plus a TPM, plus other functions (the Support Services, or SS) that do not have to be trusted to function properly. Those other functions must still operate properly if the Subsystem is to operate properly, but any misbehavior of the SS can be detected. Any misbehavior of the functions in a root, or in the TPM, on the other hand, cannot be detected.

It is not the intention of this specification to specify the method of construction of either the Subsystem or the TPM, provided that they meet the requirements of this specification. The following diagram is an indication of the functional elements of a typical TPM.

*End of informative comment.*

### 2.2.1 Definitions

**Root of Trust for Measurement (RTM)**

The point from which all trust in the measurement process is predicated. The RTM contains many components to provide this level of trust. The design document shows that the RTM includes a core component, the computing engine to run the core component, physical connections of the core and the computing engine and other items.

**Core Root of Trust for Measurement (CRTM)**

The component of the RTM from which the platform begins execution of its trusted state.

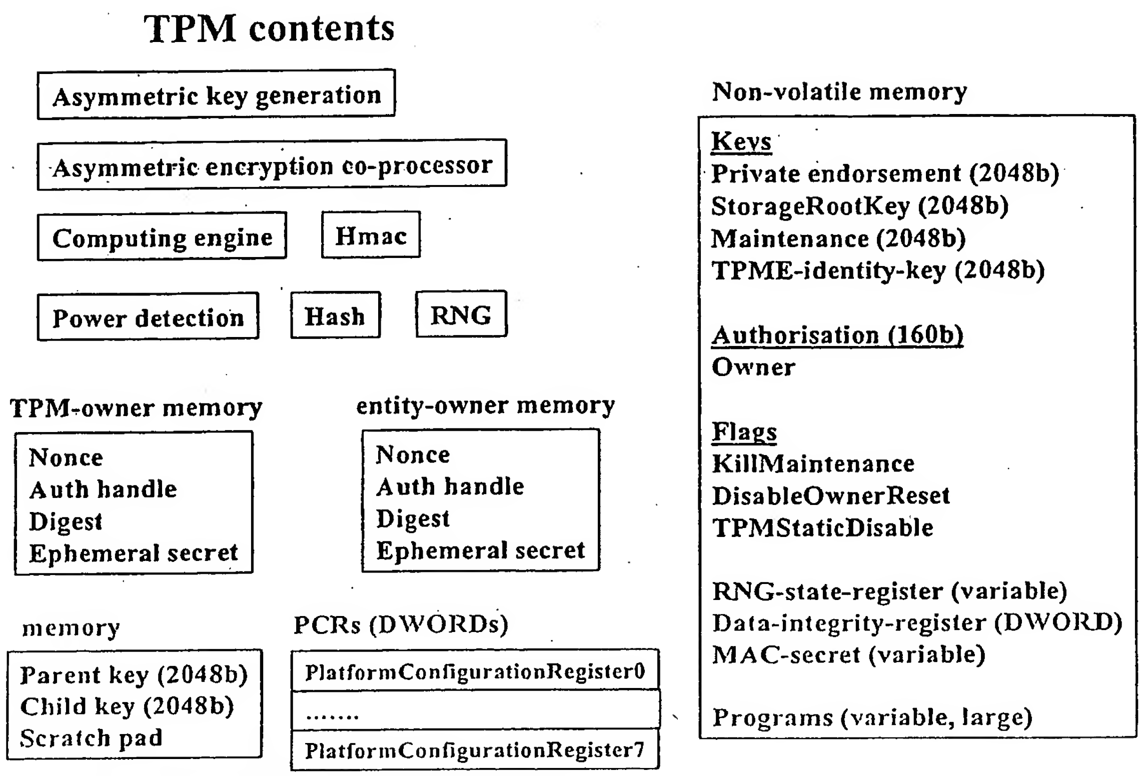
**Root of Trust for Reporting (RTR)**

The point from which all trust in reporting of measured information is predicated.

**Root of Trust for Storing (RTS)**

The point from which all trust in Protected Storage is predicated.

### 2.2.2 Instantiations and Trust Bindings




A Trusted Platform SHALL include the following:

- at least one root of trust for measuring integrity metrics,
- exactly one root of trust for storing and reporting integrity metrics,
- at least one Trusted Platform Measurement Store,
- at least one TCPA Validation Data, and
- exactly one Trusted Platform Agent.

The Endorsement Key is transitively bound to the Platform via the TPM as follows:

1. An Endorsement Key is bound to one and only one TPM (i.e., there is a one to one correspondence between an Endorsement Key and a TPM.)

2. A TPM is bound to one and only one Platform. (i.e., there is a one to one correspondence between a TPM and a Platform.)

3. Therefore, an Endorsement Key is bound to a Platform. (i.e., there is a one to one correspondence between an Endorsement Key and a Platform.)

An instantiation of the root of trust for measuring integrity metrics, while acting as the root of trust for measuring integrity metrics, SHALL do the following:

- execute no programs other than those intended by the entity that vouches for the root of trust for measuring integrity metrics,
- be resistant to the forms of software attack and to the forms of physical attack implied by the platform's Protection Profile,
- accurately measure at least one integrity metric that indicates the software environment of a platform,

- accurately record measured integrity metrics to a root of trust for storing and reporting integrity metrics, and
- accurately record details of the process of measuring all its integrity metrics to a Trusted Platform Measurement Store.

An instantiation of the root of trust for storing and reporting integrity metrics SHALL do the following:

- be resistant to all forms of software attack and to the forms of physical attack implied by the platform's Protection Profile,
- accept recording of measured integrity metrics, and
- supply an accurate digest of all sequences of presented integrity metrics.

An instantiation of a Trusted Platform Measurement Store SHOULD do the following:

- accurately accept, store and supply details of at least one process of measuring an integrity metric.

An instantiation of the repository for TCPA Validation Data SHOULD do the following:

- accurately store and supply a predicted value of at least one integrity metric.

An instantiation of the Trusted Platform Agent SHOULD do the following:

- obtain and supply an accurate report from the root of trust for storing and reporting integrity metrics of at least one sequence of integrity metrics in a form that prevents misrepresentation of that sequence or its source,
- obtain and supply an accurate report from a Trusted Platform Measurement Store of at least one set of details describing the measurement of an integrity metric, and
- obtain and supply an accurate report from the repository for TCPA Validation Data of at least one predicted value of an integrity metric

## 2.3 Integrity Operations

### 2.3.1 Storage of Integrity Metrics

*Start of informative comment:*

This section introduces the way that sequences of values of integrity metrics are stored in a TPM. This section does not describe the way that logs of the measurement process are stored in the Trusted Platform Measurement Store.

Each entry in the log inside the Trusted Platform Measurement Store contains a description of a measured entity plus an appropriate integrity metric that has been recorded inside a TPM. The log can be used to reproduce the value of each sequence of integrity metrics inside the TPM. If the log and the TPM are consistent and the TPM is trustworthy, the log can be trusted. If the values derived from the log and the values reported by the TPM are the same, the log is presumed to be an accurate record of the steps involved in building the software environment of the target platform. Consequently, the descriptions in the log of the measured entities represent the actual entities that contributed to the software environment inside the platform. Any difference between the values derived from the log and the values reported by the TPM indicate an undesirable inconsistency in the state of the target platform.

The mechanism used by the TPM to store sequences of values of integrity metrics is the subject of this section. This method must be reproduced when verifying the consistency of the values derived from the log and the values reported by the TPM.

A large number of integrity metrics may be measured in a platform, and a particular integrity metric may change with time and a new value may need to be stored. It is difficult to authenticate the source of measurement of integrity metrics, and as a result a new value of an integrity metric cannot be permitted to simply overwrite an existing value. (A rogue could erase an existing value that indicates subversion and replace it with a benign value.) Thus, if values of integrity metrics are individually stored, and updates of integrity metrics must be individually stored, it is difficult to place an upper bound on the size of memory that is required to store integrity metrics.

The TCPA solution is not to store individual integrity metrics. Instead, a Trusted Platform provides a way to store sequences of integrity metrics. Values of integrity metrics cannot be stored inside a TPM and must instead be appended to a sequence. The states of all sequences inside a TPM are set to a known value at power-up. Each new integrity metric must be appended to a sequence and must modify the value of that sequence. The actual TCPA method is to concatenate the value of a new integrity metric with the existing value of the sequence, compute a digest of the concatenation, and use that digest as the new representation of the sequence.

This method enables one or more sequences to represent an arbitrary number of integrity metrics and their updates. The fewer the number of sequences, the more difficult it becomes to interpret the meaning of the value of a sequence. The greater the number of sequences, the more costly it becomes to provide storage. A particular implementation must make a trade-off between cost and difficulty of interpretation.

***End of informative comment.***

Integrity metrics that are presented to a TPM SHALL be stored inside that TPM in a way that prevents misrepresentation of the presented values or of the sequence in which they were presented.

### 2.3.2 Reporting of Integrity Metrics

***Start of informative comment.***

This section introduces the way that sequences of integrity metrics are reported by a TPM.

An entity seeking to know the state of the computing environment inside a Trusted Platform depends critically on the values of the integrity metrics. The integrity metrics enable an entity to determine the consistency of the measurement information and compare the actual and expected states of the platform.

It follows, then, that the integrity metrics must be reported by a trusted mechanism. That trusted mechanism is the TPM (which includes the root of trust for storing and reporting integrity metrics). The TPM proclaims its trustworthiness by signing data, using one of its identities and conventional cryptographic techniques. The signature key is known only to the TPM and is the private key of a key pair. The corresponding public key is an identity key, since it is a cryptographic value by which the TPM is known. Together, the signature key and the identity key are part of an identity of the TPM.

A person or (more probably) an organization vouches for the TPM by attesting to a TPM identity. Before agreeing to provide attestation, the organization checks the construction credentials of the TPM, the design credentials of the platform that incorporates the TPM, and the construction credentials of the platform that incorporates the TPM. When the TPM reports the values of the sequences of integrity metrics that it has stored, the TPM signs those values using a TPM identity. When an entity receives signed data that originated in a TPM, the entity can verify that the data has not been changed in transit. The entity can also check that the data was signed by a TPM identity and that an organization known to the entity has attested to the TPM identity.

The TPM uses a conventional method to defeat replay attacks. That is, the entity provides a nonce that the TPM concatenates with the sequence values, before signing the values, and the signed result is returned by the Trusted Platform Agent to the entity. The actual capability provided by the TPM may be considered to be an "integrity signature". The TPM accepts arbitrary data, concatenates that arbitrary data with the sequence values, and signs the concatenated data using the signature key of a TPM

identity. When providing sequence values, that arbitrary data is simply a nonce that was provided by the challenging entity. The signed data proves that the sequence values have been supplied by a "live" TPM.

At other times, the challenging entity may wish to obtain specific information from a Trusted Platform. Then, the arbitrary data could be a digest of the specific information. The signed data proves the state of the computing environment inside the Trusted Platform at the time that the specific information was supplied.

*End of informative comment.*

Sequences of integrity metrics reported by the TPM SHALL be reported by that TPM in a way that prevents misrepresentation of the sequences and prevents misrepresentation of the reporting TPM

## 2.4 Use of Keys Associated with TPM Identities

*Start of informative comment.*

The private key associated with a TPM identity is used only for signatures. Such signatures lend credibility to signed data, because the data must have been signed by a TPM.

The private keys associated with TPM identities must be indelibly stored with flags that mark them as belonging to TPM identities, in order that they can be distinguished from other types of keys. This is necessary to enforce restrictions on the use of those keys.

TPM identities can be used to sign certain data, and a TPM must refuse to use private keys associated with TPM identities for other purposes. Otherwise, a rogue may construct data (outside the TPM) that has the same format as that used by the TPM for special operations, and cause a TPM to sign that data using a private key associated with TPM identity. Such data would be misinterpreted as genuine data constructed by the TPM for those special purposes, and could subvert the trust in those special purposes. If the TPM prevents such a masquerade, a third party can always be certain that data (signed by a private key associated with a TPM identity) was actually generated by a TPM for one of those special operations.

*End of informative comment.*

It MUST be possible to reliably distinguish between the private key of a TPM identity and other keys.

A key that is distinguished as the private key of a TPM identity SHALL NOT be used to generate a digital signature value over data that could mimic the output of a TCPA protected capability.

A TPM SHALL NOT use a key that is distinguished as the private key of a TPM identity except during the part of a TCPA "protected capability" whose specification permits and/or requires the use of a TPM identity.

When signing on behalf of a TPM identity during the part of a TCPA protected capability whose specification requires the signature of a TPM identity, a TPM SHALL NOT use a key other than one that is distinguished as the private key of a TPM identity.

## 2.5 Cryptographic Operations

*Start of informative comment.*

This section introduces the use of cryptographic operations within the Subsystem. Note that this specification does not include the AES. It is probable, however, that future versions of this specification will include the AES.

The Subsystem employs conventional cryptographic operations in conventional ways. Those operations include the following:

- Hashing (SHA-1)
- Random number generation (RNG)
- Asymmetric key generation (RSA)

- Asymmetric encryption/decryption (RSA)
- Symmetric encryption/decryption (3DES)

The Subsystem uses these capabilities to perform generation of random data, generation of asymmetric and symmetric keys, signing and confidentiality of stored data. The Subsystem also uses confidential messaging for its own purposes, but does not provide a general-purpose symmetric confidentiality service. This choice is deliberate, because the fundamental TCPA objective is to improve trust in a general-purpose computing platform. Hence, TCPA provides only those functions that are necessary to improve confidence in such a platform so that processing (including conventional security functions) on the platform can be done with greater confidence.

The TPM contains the minimum set of capabilities that are required to be trusted. The TPM capabilities must be trustworthy if the Subsystem is to be trusted. Other Subsystem capabilities must (of course) function properly if the Subsystem is to work as expected.

The TPM contains the following crypto capabilities:

- Hashing (SHA-1)
- Random number generation (RNG)
- Asymmetric key generation (RSA)
- Asymmetric encryption/decryption (RSA)

Note that this list does not include symmetric encryption. This is for reasons of cost.

The hash capability is for use primarily by the TPM, since the TPM requires access to a trusted hash function. The hash capability is exported by the TPM just to improve hash availability during the boot phase of a platform, when the RTM and other measurement agents probably have restricted access to the platform's main processing engine.

The untrusted part of the Subsystem must include symmetric encryption functionality, but does not include an RNG. The TSS may also include duplicate asymmetric key generation and asymmetric encryption capabilities depending on the usefulness of TCPA protected capabilities to the TSS.

The Random Number Generator consists of a state-machine that accepts and mixes unpredictable data and a post-processor that is a one-way function (such as a hash algorithm). This architecture is chosen to provide a good source of random data without requiring that the TPM include a genuine source of unpredictable data (which may be expensive).

The state-machine has non-volatile state, is initialized with unpredictable data before delivery to a customer, and can at any time accept further (unpredictable) data. Such data may be provided by hardware (from thermal noise, for example), or by software (monitoring keyboard strokes, for example). Some such unpredictable data must be inserted every time that a platform boots. Naturally, a hardware source is likely to supply data at a higher baud rate than a software source. That further data is mixed into the existing state of the machine and as a result improves the unpredictability of the state of the state-machine. Neither the Owner of the TPM nor the manufacturer of the TPM can deduce the state of the state-machine. The post-processor is used to "condense" the output of the state-machine into data that has sufficient and uniform entropy. (The one-way function will use more bits of input data than it produces as output.)

*End of informative comment.*

## 2.6 Opting to use a TPM

*Start of informative comment.*

It is necessary to provide features that activate a TPM. This is for reasons of privacy.

A TPM is necessarily activated by a reset. This, however, causes the TPM to discard any existing secrets, and puts the TPM into its virgin state, waiting for an Owner. It leaves the TPM vulnerable to ownership by

anyone who knows the PUBEK of the TPM and can get a "take ownership" command to the TPM. To fail safe, the true Owner would need to take ownership as soon as possible after a TPM has been reset. If desired, the true Owner could then withhold the authorization information that is necessary to use the TPM. Since a TPM can have only one Owner, this prevents any use of the TPM until the true Owner decides to use it.

It is therefore desirable to provide methods that deactivate and activate a TPM without destroying existing secrets. Then the Owner of the TPM (or a user) may deactivate the TPM in order to prevent inadvertent use of the TPM, and later reactivate the TPM in order to use current secrets. It is also desirable to provide methods that activate and deactivate the process of taking ownership, in case the true Owner does not wish to take ownership (at least, not yet).

The TCPA specification defines a set of capabilities to enable/disable a TPM, activate/deactivate a TPM and enable/disable the process of taking ownership of the TPM.

The overall effect of the disabling capabilities is that a disabled TPM does little of value, apart from keeping accurate records of integrity metrics and acknowledging that the TPM exists. A disabled TPM is therefore, effectively, "off".

The overall effect of the deactivating capabilities is that an inactive TPM does nothing, apart from keeping accurate records of integrity metrics, acknowledging that the TPM exists, and permitting the process of installing an owner in the TPM.

There are obviously many combinations of the particular states of TPM enabled/disabled, TPM active/inactive, install-owner enabled/disabled. It may be that some suppliers will choose to supply a virgin TPM that is enabled, active, and with install-owner enabled, because that is what is required by their customer. At the other extreme, if a virgin TPM is supplied in the disabled and inactive state, with "take ownership" disabled, three steps are required in order to activate the TPM. One possible activation sequence would be:

1. The prospective Owner should enable the TPM
2. The prospective Owner should attempt to take ownership
3. The prospective Owner should activate the TPM

This particular sequence gives maximum control to the Owner, and permits verification that taking ownership has succeeded, before the TPM is activated.

There are other possibilities between these two extremes. It may be that a virgin TPM is enabled but inactive, with "take ownership" disabled, for example. This may be an advantage if the process of enabling a TPM is non-trivial.

***End of informative comment.***

### 2.6.1 Enabling Ownership

***Start of informative comment:***

If a TPM does not have an Owner, it is desirable to provide a method that enables or disables the process by which a prospective Owner takes ownership of a TPM. Ideally this method would work both locally and remotely. Unfortunately authenticated commands cannot be interpreted by the TPM if it does not have an Owner. Hence the method of enabling or disabling the process of taking ownership is a local command and no remote option is provided. (In a PC, these local controls could be made available during the POST, for example.)

***End of informative comment.***

### 2.6.2 Activating a TPM

***Start of informative comment:***

It is desirable to provide methods that activate or deactivate a TPM without permanently preventing access to secrets protected by the TPM. The provision of deactivation methods exposes a denial-of-service attack, but this is considered a worthwhile price to pay for improved privacy.

One method should certainly be the use of commands authorized by the Owner. This method has the advantage that it proves possession of sufficient privilege, and can be used either locally or remotely. A drawback of this method is that the platform must (probably) be fully active in order to communicate an authorized command to a TPM. The concern is that the TPM may inadvertently be used inbetween the platform becoming fully active and an authorized "deactivate" command being received by the TPM. Another disadvantage is that it may be necessary to disable a TPM when the Owner is not available. Other methods are, therefore, also required. The scope of these methods must reflect any uncertainty about possession of sufficient privilege.

One method is required to operate before the platform is fully active. In these circumstances, it may be difficult to check authorization. The method adopted by TCPA is to use software controls that are remotely inaccessible. These are intended to provide local activation only (not remote activation), but this depends upon the degree to which the control software is actually inaccessible to remote entities.

Another method is to required to operate when the platform is fully active, but without Owner authorization. The method adopted by the TCPA is to use an unauthorized command that has a limited effect – it can be used just to deactivate a TPM and the effect lasts only until the platform is rebooted.

The method of final resort to activate a TPM is to use a physical (electrical) input to the TPM that cannot be controlled by software executing on the main platform. This method (obviously) provides local activation but not remote activation. This method is useful if no one has taken ownership, or the Owner's authorization has been lost, but one or more User authorization data are still known. In the latter case, the TPM can be activated and Users can use their secrets to recover as much as possible of their data.

This specification uses four methods of activation (while retaining current TPM secrets):

1. A physical (electrical) input to the TPM that cannot be controlled by software executing on the main platform. Enabling this physical input could involve opening of the platform and throwing a switch, or activation of a physical lock, for example. Each use of the control causes a transitory activate event at the TPM. This (obviously) provides local activation but not remote activation.

2. An authenticated command to the TPM from the Owner. This provides either local or remote activation of the TPM.

3. The use of software controls that are remotely inaccessible. These are intended to provide local activation and not remote activation, but that property depends upon the degree to which the controlling software is actually inaccessible to remote entities. (In a PC, these controls could be made available during the POST, for example.)

4. A power-cycle of the platform. This is intended to provide local activation and not remote activation, but that property depends upon the degree to which a reboot is actually inaccessible to remote entities.

This specification uses three methods of deactivation (while retaining current TPM secrets):

1. An authenticated command to the TPM from the Owner. These provide either local or remote deactivation of the TPM.

2. An unauthenticated command to the TPM. These provide either local or remote deactivation of the TPM.

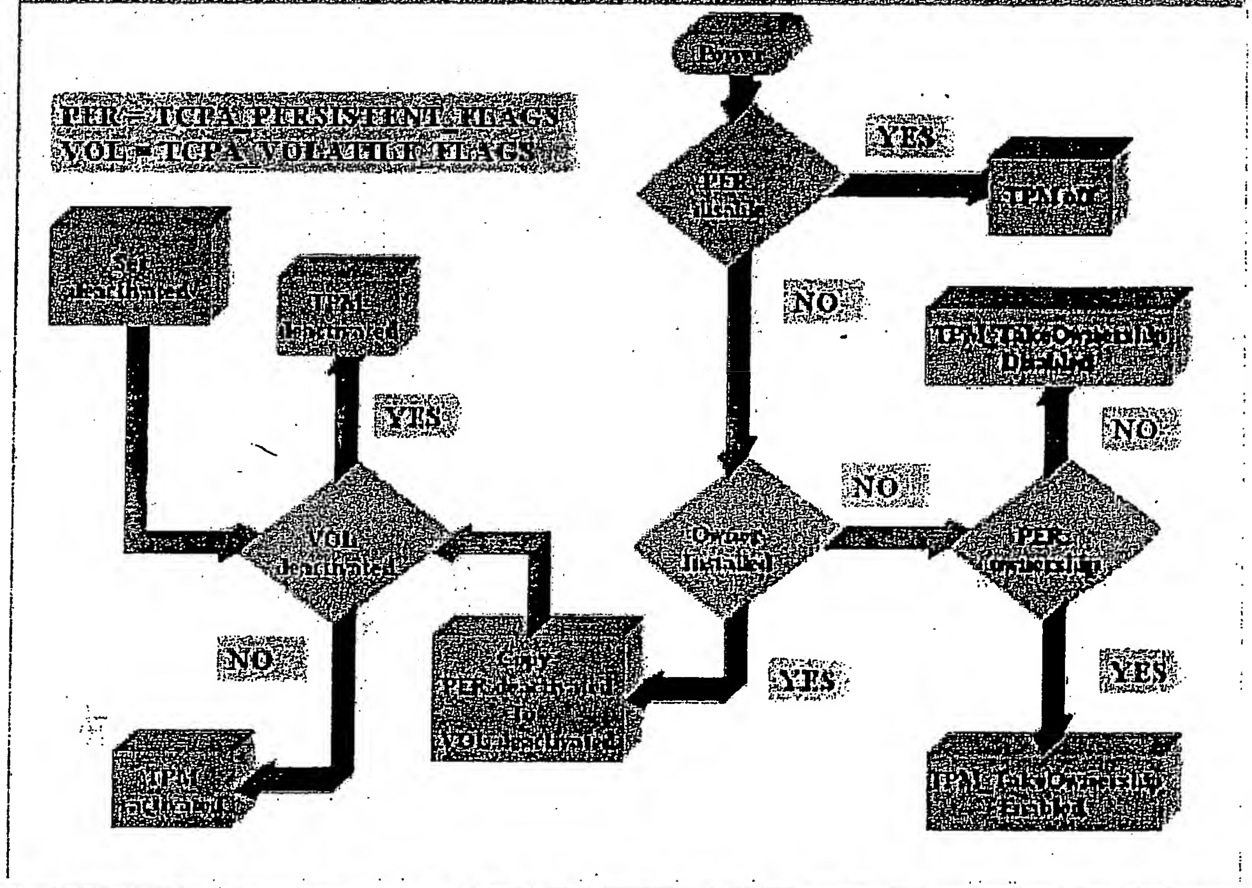
3. The use of software controls that are remotely inaccessible. These are intended to provide local deactivation and not remote deactivation, but that property depends upon the degree to which the controlling software is actually inaccessible to remote entities. (In a PC, these controls could be made available during the POST, for example.)

***End of informative comment.***

### 2.6.3 Selected operations

*Start of informative comment*

The methods to enable/disable a TPM, activate/deactivate a TPM, and enable/disable the process of taking ownership of the TPM, can be combined in many ways. The selection made by TCPA is illustrated in the following flowchart diagram, which illustrates a sequence of tests and decisions after Power-On Reset (POR).




| Bit | Flag name | Flag type | Action to set TRUE | Action to set FALSE |
|---|---|---|---|---|
| 1 | TCPA_PERSISTENT_FLAGS disable | Non-volatile | 1) Owner auth cmd<br>2) Local cmd | 1) Owner auth cmd<br>2) physical action |
| 2 | TCPA_PERSISTENT_FLAGS ownership | Non-volatile | Local cmd | Local cmd |
| 3 | TCPA_PERSISTENT_FLAGS deactivated | Non-volatile | Local cmd | Local cmd |
| 4 | TCPA_VOLATILE_FLAGS deactivated | Volatile | Unauth cmd | Platform reboot |

(BIT 1) This may be set or reset by an Owner authorized command (TPM_SetOwnerInstall 8.13.1). It may be set by a local command (TPM_PhysicalDisable 8.14.2). It may be reset by a physical action (TPM_PhysicalEnable 8.14.3).

These methods permit the Owner to disable the TPM when necessary (provided the TPM is accepting authorized commands from the Owner); permit a User or a Owner to disable a TPM via

local access to the platform, and permit a User or Owner to activate a TPM by the use of physical access to the platform (which may or may not be trivial).

The TPM is disabled by a command that has originated locally. It may be that this "local" requirement restricts the operation of this command to times before an OS is running. The TPM is also disabled by an Owner authorized command. It may be that this "authorization" requirement restricts this command to times after the OS is running.

The TPM can be enabled by a physical event at the platform (whether or not the TPM has an Owner, and whether or not the OS is running). The TPM can also be enabled by an Owner authorized command. It may be that this "authorization" requirement restricts this command to times after the OS is running.

(BIT 2) This may be set or reset by a local command (TPM_SetOwnerInstall 8.13.1).

This method permits a User or Owner to enable or disable the process of taking ownership, via local access to the platform. It may be that this "local" requirement restricts the operation of this command to times before an OS is running.

(BIT 3) This may be set or reset by a local command (TPM_PhysicalSetDeactivated 8.15.1).

This method permits a User or an Owner to set the default active/deactive state of a TPM via local access to the platform. It may be that this "local" requirement restricts the operation of these commands to times before an OS is running.

(BIT 4) This may be set by a local command (TPM_SetTempDeactivated 8.15.2). Any alteration lasts until the next boot cycle, when this bit is initialized to the state of BIT3.

This method permits a User or the Owner to temporarily deactivate the TPM. An unauthorized command causes the TPM to enter an inactive state. The TPM remains in that state until the platform is rebooted.

The default states of the persistent bits (BIT 1, 2, 3) in a virgin platform are the choice of the supplier. In a platform where "physical access" involves opening the platform, a supplier may wish to set DISABLE_TPM=FALSE, for example. In a platform where the supplier knows that the customer will use the Subsystem, a supplier may wish to set DISABLED_OWNER_INSTALL=FALSE and DEACTIVATED_TPM=FALSE, for example. In a platform where the supplier is uncertain whether the customer will use the Subsystem, a supplier may wish to set DISABLED_OWNER_INSTALL=TRUE and DEACTIVATED_TPM=TRUE, for example.

Both a disabled TPM and an inactive TPM never prevent the "extend" capability from operating. This is necessary in order to ensure that the records of sequences of integrity metrics in a TPM are always up-to-date.

***End of informative comment.***

## 2.7 Protected, Unprotected, and Connection Operations

*Start of informative comment:*

All TCPA protected capabilities are provided by the TPM. The TPM requires the TSS to properly perform its functions. The TSS by definition has NO security sensitive operations defined. Failure to properly perform a TSS function may cause a TPM operation to fail but the failure will not result in a security exposure.

TSS operations and protocols to support the TPM are defined in this specification as informative and normative statements only. More detailed aspects of those TSS operations, such as command and parameter structures, may be defined in other TCPA specifications.

Connection Operations can be defined to enable TPM Operations such as those requiring physical presence.

*End of informative comment.*

No operation outside the TPM SHALL affect the security of the TPM, only the ability of the TPM to operate. TCPA Operations are classified as:

- Protected Operations — Operations affecting the security properties of TCPA. These are TPM Operations. These begin with TPM_
- Unprotected Operations — Operations supporting the protected operations. These are normally implemented outside the TPM. This begin with TSS_
- Connection Operations — Operations affecting the connection of the platform to the TPM. These are typically defined in the Platform Specific specifications. These begin with TSC_.

# 3. Protection

## 3.1 Introduction

***Start of informative comment:***

The Protection Profile in the Conformance part of the specification defines the threats that are resisted by a platform. This section, "Protection", describes the properties of selected capabilities and selected data locations within a platform that has a Protection Profile and has not been modified by physical means.

This section introduces the concept of protected capabilities and the concept of shielded locations for data. Every definition of a TCPA capability states whether it is a protected capability. Data definitions state whether the data must be held in shielded locations.

- A protected capability is one whose correct operation is necessary in order for the operation of the Subsystem to be trusted.
- A shielded location is an area where data is protected against interference and prying, independent of its form.

This specification uses the concept of protected capabilities so as to distinguish those Subsystem capabilities that must be trustworthy. Trust in the Subsystem depends critically on the protected capabilities. Subsystem capabilities that are not protected capabilities must (of course) work properly if the Subsystem is to function properly.

This specification uses the concept of shielded locations, rather than the concept of "shielded data". While the concept of shielded data is intuitive, it is extraordinarily difficult to define because of the imprecise meaning of the word "data". For example, consider data that is produced in a safe location and then moved into ordinary storage. It is the same data in both locations, but in one it is shielded data and in the other it is not. Also, data may not always exist in the same form. For example, it may exist as vulnerable plaintext, but also may sometimes be transformed into a logically protected form. This data continues to exist, but doesn't always need to be shielded data - the vulnerable form needs to be shielded data, but the logically protected form does not. If a specific form of data requires protection against interference or prying, it is therefore necessary to say "if the data-D exists, it must exist only in a shielded location". A more concise expression is "the data-D must be extant only in a shielded location".

Hence if trust in the Subsystem depends critically on access to certain data, that data should be extant only in a shielded location and accessible only to protected capabilities. When not in use, such data could be erased after conversion (using a protected capability) into another data structure. Unless the other data structure was defined as one that must be held in a shielded location, it need not be held in a shielded location.

***End of informative comment.***

## 3.2 Threat

***Start of informative comment:***

This section, "Threat", defines the scope of the threats that must be considered when considering whether a platform facilitates subversion of capabilities and data in a platform.

The design and implementation of a platform determines the extent to which the platform facilitates subversion of capabilities and data within that platform. It is necessary to define the attacks that must be resisted by TCPA-shielded locations and TCPA-protected capabilities in that platform.

The TPM Protection Profile defines all attacks that are resisted by the TPM. These attacks must be considered when determining whether the integrity of TCPA-protected capabilities and data in TCPA-shielded locations can be damaged. These attacks must be considered when determining whether there is a backdoor method of obtaining access to TCPA-protected capabilities and data in TCPA-shielded locations. These attacks must be considered when determining whether TCPA-protected capabilities have undesirable side effects.

*End of informative comment*

For the purposes of the "Protection" section of the specification: the threats that MUST be considered when determining whether the platform facilitates subversion of TCPA-protected capabilities or data in TCPA-shielded locations SHALL include the methods inherent in physical attacks that should fail if the platform complies with its protection profile, and SHALL include all methods that require execution of instructions in a computing engine in the platform.

## 3.3 Integrity

*Start of informative comment:*

A TCPA-protected capability must be used to modify TCPA-protected capabilities or data in TCPA-shielded locations. Other methods must not be allowed to modify TCPA-protected capabilities or data in TCPA-shielded locations. Otherwise, the integrity of TCPA-protected capabilities and data in TCPA-shielded locations is unknown.

*End of informative comment*

A platform SHALL NOT facilitate the alteration of TCPA-protected capabilities or data in TCPA-shielded locations, except by TCPA-protected capabilities.

## 3.4 Privileged Access

*Start of informative comment:*

Only TCPA-protected capabilities are allowed to use the data in TCPA-shielded locations. Otherwise, a rogue can pretend to be a TCPA entity.

*End of informative comment*

A platform SHALL NOT facilitate the disclosure or the exposure of data in TCPA-shielded locations, except to TCPA-protected capabilities.

## 3.5 Side effects

*Start of informative comment:*

An implementation of a TCPA-protected capability must not disclose the contents of TCPA-shielded locations. The only exceptions are when such disclosure is inherent in the definition of the capability or in the methods used by the capability. For example, a capability might be designed specifically to reveal hidden data or might use cryptography and hence always be vulnerable to cryptanalysis. In such cases some disclosure or risk of disclosure is inherent and cannot be avoided. Other forms of disclosure (by side effects, for example) must always be avoided.

*End of informative comment*

The implementation of a TCPA-protected capability in a platform SHALL NOT facilitate the disclosure or the exposure of data in TCPA-shielded locations except by means unavoidably inherent in the TCPA definition.

# 4. Structures and Defines

*Start of informative comment:*

The following structures and formats describe the interoperable areas of the specification. There is no requirement that internal storage or memory representations of data must follow these structures. These requirements are in place only during the movement of data from a TPM to some other entity.

*End of informative comment.*

### 4.1.1 Endness of Structures

Each structure MUST use big endian bit ordering, which follows the Internet standard and requires that the low-order bit appear to the far right of a word, buffer, wire format, or other area and the high-order bit appear to the far left.

### 4.1.2 Byte Packing

All structures MUST be packed on a byte boundary.

### 4.1.3 Lengths

The "Byte" is the unit of length when the length of a parameter is specified.

## 4.2 Defines

*Start of informative comment:*

The defines are found in tcpa_defines.h

*End of informative comment.*

### 4.2.1 Basic data types

**Parameters**

| Typedef | Name | Description |
|---|---|---|
| unsigned char | BYTE | Basic byte used to transmit all character fields. |
| unsigned char | BOOL | TRUE/FALSE field. TRUE = 0x01, FALSE = 0x00 |
| unsigned short | UINT16 | 16 bit field. The definition in different architectures may need to specify 16 bits instead of the short definition |
| unsigned long | UINT32 | 32 bit field. The definition in different architectures may need to specify 32 bits instead of the long definition |

### 4.2.2 Boolean types

| Name | Value | Description |
|---|---|---|
| TRUE | 0x01 | Assertion |
| FALSE | 0x00 | Contradiction |

### 4.2.3 Helper redefinitions

The following definitions are to make the IDL definitions more explicit and easier to read.

**Parameters**

| Typedef | Name | Description |
|---|---|---|
| UINT32 | TCPA_PCRINDEX | Index to a PCR register |
| UINT32 | TCPA_DIRINDEX | Index to a DIR register |
| UINT32 | TCPA_AUTHHANDLE | Handle to an authorization session |
| UINT32 | TSS_HASHHANDLE | Handle to a hash session |
| UINT32 | TSS_HMACHHANDLE | Handle to a HMAC session |
| UINT32 | TCPA_ENCHANDLE | Handle to a encryption/decryption session |
| UINT32 | TCPA_KEY_HANDLE | The area where a key is held assigned by the TPM. |
| UINT32 | TCPA_RESULT | The return code from a function |

### 4.2.4 Enumerated Helper redefinitions

| Typedef | Name | Description |
|---|---|---|
| UINT32 | TCPA_COMMAND_CODE | The command ordinal. See 4.33 |
| UINT16 | TCPA_PROTOCOL_ID | The protocol in use. See 4.17 |
| UINT32 | TCPA_EVENTTYPE | Type of PCR event. See 4.25.2 |
| BYTE | TCPA_AUTH_DATA_USAGE | Indicates the conditions where it is required that authorization be presented. See 4.11 |
| UNIT16 | TCPA_ENTITY_TYPE | Indicates the types of entity that are supported by the TPM. See 4.15 |
| UNIT32 | TCPA_ALGORITHM_ID | Indicates the type of algorithm. See 4.18 |
| UNIT16 | TCPA_KEY_USAGE | Indicates the permitted usage of the key. See 4.10 |
| UINT16 | TCPA_STARTUP_TYPE | Indicates the start state. See 4.16 |
| UINT32 | TCPA_CAPABILITY_AREA | Identifies a TPM capability area. See 4.31 |
| UINT16 | TCPA_ENC_SCHEME | The definition of the encryption scheme. See 8.4 |
| UINT16 | TCPA_SIG_SCHEME | The definition of the signature scheme. See 8.5 |
| UINT16 | TCPA_MIGRATE_SCHEME | The definition of the migration scheme 4.22 |
| UINT16 | TCPA_PHYSICAL_PRESENCE | Sets the state of the physical presence mechanism. See section 4.19 |
| UINT32 | TCPA_KEY_FLAGS | Indicates information regarding a key. See 4.12 |

### 4.2.5 Vendor specific

*Start of informative comment:*

For all items that can specify an individual algorithm, protocol or item the specification allows for vendor specific selections. The mechanism to specify a vendor specific mechanism is to set the high bit of the identifier on.

*End of informative comment.*

The following defines allow for the quick specification of a vendor specific item.

**Parameters**

| Name | Value |
|---|---|
| TCPA_Vendor_Specific32 | 0x80000000 |
| TCPA_Vendor_Specific16 | 0x8000 |
| TCPA_Vendor_Specific8 | 0x80 |

## 4.3 Return codes

*Start of informative comment:*

The TPM has two types of returns, TCPA_SUCCESS where the TPM reports the results of a successful function execution and the failure return.

The failure case only returns a non-authenticated fixed set of information. This is due to the fact that the failure may have been due to authentication or other factors and there is no possibility of producing an authenticated response.

Failures also terminate any authorization sessions. This is a result of returning only the error code as there is no way to return and continue the nonces necessary to maintain an authorization session.

*End of informative comment.*

**Description**

When a command fails for ANY reason, the TPM MUST return only the following three items:

- TPM_TAG_RQU_COMMAND (2 bytes)
- ParamLength(4 bytes, fixed at 10)
- Return Code (4 bytes, never TCPA_SUCCESS)

If a return code is mandated by the action list of a command the TPM MUST return that error code. All commands MAY return TPM_FAIL, where there is a more descriptive error code the TPM SHOULD use the more descriptive error code.

The return code MUST be chosen from the following list.

**Parameters**

| Name | Value | Description |
|---|---|---|
| TCPA_BASE | 0x0 | The start of TCPA return codes |
| TCPA_SUCCESS | TCPA_BASE | Successful completion of the operation |
| TCPA_VENDOR_ERROR | TCPA_BASE + TCPA_Vendor_Specific32 | These error codes are vendor specific for vendor specific commands. |
| TCPA_AUTHFAIL | TCPA_BASE + 1 | Authentication failed |
| TCPA_BADINDEX | TCPA_BASE + 2 | The index to a PCR, DIR or other register is incorrect |
| TCPA_BAD_PARAMETER | TCPA_BASE + 3 | One or more parameter is bad |
| TCPA_AUDITFAILURE | TCPA_BASE + 4 | An operation completed successfully but the auditing of that operation failed. |
| TCPA_CLEAR_DISABLED | TCPA_BASE + 5 | The clear disable flag is set and all clear operations now require physical access |
| TCPA_DEACTIVATED | TCPA_BASE + 6 | The TPM is deactivated |
| TCPA_DISABLED | TCPA_BASE + 7 | The TPM is disabled |
| TCPA_DISABLED_CMD | TCPA_BASE + 8 | The target command has been disabled |
| TCPA_FAIL | TCPA_BASE + 9 | The operation failed |
| TCPA_INACTIVE | TCPA_BASE + 10 | The TPM is inactive |

| | | |
|---|---|---|
| TCPA_INSTALL_DISABLED | TCPA_BASE + 11 | The ability to install an owner is disabled |
| TCPA_INVALID_KEYHANDLE | TCPA_BASE + 12 | The key handle presented was invalid |
| TCPA_KEYNOTFOUND | TCPA_BASE + 13 | The target key was not found |
| TCPA_NEED_SELFTEST | TCPA_BASE + 14 | The capability requires an untested function; additional self-test is required before the capability may execute. |
| TCPA_MIGRATEFAIL | TCPA_BASE + 15 | Migration authorization failed |
| TCPA_NO_PCR_INFO | TCPA_BASE + 16 | A list of PCR values was not supplied |
| TCPA_NOSPACE | TCPA_BASE + 17 | No room to load key. |
| TCPA_NOSRK | TCPA_BASE + 18 | There is no SRK set |
| TCPA_NOTSEALED_BLOB | TCPA_BASE + 19 | An encrypted blob is invalid or was not created by this TPM |
| TCPA_OWNER_SET | TCPA_BASE + 20 | There is already an Owner |
| TCPA_RESOURCES | TCPA_BASE + 21 | The TPM has insufficient internal resources to perform the requested action. |
| TCPA_SHORTRANDOM | TCPA_BASE + 22 | A random string was too short |
| TCPA_SIZE | TCPA_BASE + 23 | The TPM does not have the space to perform the operation. |
| TCPA_WRONGPCRVAL | TCPA_BASE + 24 | The named PCR value does not match the current PCR value. |
| TCPA_BUSY | TCPA_BASE + 25 | The TPM is too busy to respond to the command |
| TCPA_SHA_THREAD | TCPA_BASE + 26 | There is no existing SHA-1 thread. |
| TCPA_SHA_ERROR | TCPA_BASE + 27 | The calculation is unable to proceed because the existing SHA-1 thread has already encountered an error. |
| TCPA_FAILEDSELFTEST | TCPA_BASE + 28 | Self-test has failed and the TPM has shutdown. |
| TCPA_AUTH2FAIL | TCPA_BASE + 29 | The authorization for the second key in a 2 key function failed authorization |
| TCPA_BADTAG | TCPA_BASE + 30 | The tag value sent to for a command is invalid |
| TCPA_IOERROR | TCPA_BASE + 31 | An IO error occurred transmitting information to the TPM |
| TCPA_ENCRYPT_ERROR | TCPA_BASE + 32 | The encryption process had a problem. |
| TCPA_DECRYPT_ERROR | TCPA_BASE + 33 | The decryption process did not complete. |
| TCPA_INVALID_AUTHHANDLE | TCPA_BASE + 34 | The auth handle was invalid |
| TCPA_NO_ENDORSEMENT | TCPA_BASE + 35 | The TPM does not a EK installed |
| TCPA_INVALID_KEYUSAGE | TCPA_BASE + 36 | The usage of a key is not allowed |
| TCPA_WRONG_ENTITYTYPE | TCPA_BASE + 37 | The submitted entity type is not allowed |
| TCPA_INVALID_POSTINIT | TCPA_BASE + 38 | The command was received in the wrong |

| | | |
|---|---|---|
| | | sequence relative to TPM_Init and a subsequent TPM_Startup |

## 4.4 Command Specification Table Description

### 4.4.1 Introduction, Definition of Terms

- The parameter order column (*PARAM)* lists the order in which the parameters must be added to the input or output array and their respective size. If this entry in the column is blank, then that parameter is not sent to the TPM driver.
- <> in size column means that the size of the element is defined by the appropriate input parameter (sizeInData controls inData). Where an explicit input 'size' parameter exists, it has been moved to immediately precede the array to which it refers so that there is no confusion.
- When a null terminated string is included in a calculation, the terminating null SHALL NOT be included in the calculation.
- The following rules concerning byte ordering within a parameter are consistent with Section 4.1 and follow Internet standards:
    1. Elements of a structure are marshaled in the order in which they appear in the document.
    2. Byte arrays are marshaled starting with index 0, followed by index 1, and so on.
    3. Integer types are marshaled most significant byte first.
    4. No padding bytes are to be inserted at any point.
    5. Bit ordering within the byte is determined by the IO channel in use.
- Parameters are marshaled into the input or output arrays according to the following order:
    1. Tag specifier
    2. Array length, including tag and length specifier bytes
    3. Command ordinal and/or return code
    4. Key handles
    5. Remaining fixed length parameters
    6. Remaining variable length parameters (with their size parameter)
    7. If applicable, First authorization setup (authHandle – input only, then nonce, then continueUse)
    8. If applicable, First Authorization digest
    9. If applicable, Second authorization setup
    10. If applicable, Second authorization digest

### 4.4.2 HMAC Calculation for Authorization

- All authorized parameters other than the authorization setup parameters (authHandle, nonces and continueUse) are hashed using SHA-1. This digest, referred to as <paramDigest> throughout this document, is HMAC'd with the authorization setup parameters to form the authorization digest.
- Where there are two authorization sessions within a single command (changeAuth, etc.) the two HMACs are computed using the common <paramDigest> but their respective setup parameters only.
    1. AuthDigest1 = HMAC( <paramDigest>, EvenNonce1, OddNonce1, continueUse1)
    2. AuthDigest2 = HMAC( <paramDigest>, EvenNonce2, OddNonce2, continueUse2)
- The comment after the HMAC authorization digest includes the source of the HMAC key for the digest. If the authorization session is of type OSAP, then the actual key is the sharedSecret that was

derived from the secret listed in the comment. For OIAP sessions, the HMAC key is the listed secret directly.

- In the tables below, the order of computation of the SHA1 hash and HMACs are shown in the HMAC column. The subscript 'S' refers to parameters that are hashed together using SHA1 to form <paramDigest>. The subscripts 'H1' & 'H2' refer to parameters that are HMAC'd to form the first and second authorization digests.
- Note that as the first element to the HMAC calculation is <paramDigest>, HMAC element numbers start with 2 in all cases below.
- In all cases, both input and output, the HMAC calculation uses the following order:
    1. <paramDigest>
    2. Even nonce (generated by TPM)
    3. Odd nonce (generated by system)
    4. ContinueUse

### 4.4.3 Parameter List Tag Identifiers

| Tag | Name | Description |
|---|---|---|
| 0x00C1 | TPM_TAG_RQU_COMMAND | A command with no authentication. |
| 0x00C2 | TPM_TAG_RQU_AUTH1_COMMAND | An authenticated command with one authentication handle |
| 0x00C3 | TPM_TAG_RQU_AUTH2_COMMAND | An authenticated command with two authentication handles |
| 0x00C4 | TPM_TAG_RSP_COMMAND | A response from a command with no authentication |
| 0x00C5 | TPM_TAG_RSP_AUTH1_COMMAND | An authenticated response with one authentication handle |
| 0x00C6 | TPM_TAG_RSP_AUTH2_COMMAND | An authenticated response with two authentication handles |

## 4.5 TCPA_VERSION

*Start of informative comment:*

The TCPA_VERSION allows the TPM to communicate with outside entities as to the version of the TPM. This structure is set by the TPM and included in structures that are maintained long term outside of the TPM.

*End of informative comment.*

**IDL Definition**

```
typedef struct tdTCPA_VERSION {
 BYTE major;
 BYTE minor;
 BYTE revMajor;
 BYTE revMinor;
} TCPA_VERSION;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| BYTE | major | This SHALL be the major version indicator. For version 1 this MUST be 0x01 |
| BYTE | minor | This SHALL be the minor version indicator. For version 1 this MUST be 0x01 |
| BYTE | revMajor | This SHALL be the value of the TCPA_PERSISTENT_DATA -> revMajor |
| BYTE | revMinor | This SHALL be the value of the TCPA_PERSISTENT_DATA -> revMinor |

**Descriptions**

The version points to the version of the specification that defines the structure.

If the validity of a structure depends on conformity to a version of the specification and/or to a version of the TPM, that structure SHALL include the current instance of TCPA_VERSION

## 4.6 TCPA_DIGEST

*Start of informative comment*

The digest value reports the result of a hash operation. In Version 1.0 of this specification the hash algorithm is SHA-1 with a resulting hash result being 160 bits. This lack of flexibility is because the size of a digest has a dramatic effect on the implementation of a hardware TPM.

*End of informative comment*

**Definition**

```
typedef struct tdTCPA_DIGEST{
    BYTE digest[digestSize];
} TCPA_DIGEST;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| BYTE | digest | This SHALL be the actual digest information |

**Description**

The digestSize parameter MUST indicate the block size of the algorithm and MUST be 20 or greater.

For all TCPA v1 hash operations, the hash algorithm MUST be SHA-1 and the digestSize parameter is therefore equal to 20.

**Redefinitions**

| Typedef | Name | Description |
|---|---|---|
| TCPA_DIGEST | TCPA_PCRVALUE | The value inside of the PCR |
| TCPA_DIGEST | TCPA_COMPOSITE_HASH | This SHALL be the hash of a list of PCR indexes and PCR values that a key or data is bound to (See 10.4.5 for details) |
| TCPA_DIGEST | TCPA_DIRVALUE | This SHALL be the value of a DIR register |
| TCPA_DIGEST | TCPA_HMAC | |
| TCPA_DIGEST | TCPA_CHOSENID_HASH | This SHALL be the digest of the chosen identityLabel and privacyCA for a new TPM identity. See 10.4.6 for details. |

## 4.7 TCPA_NONCE

*Start of informative comment:*

A nonce is a random value that provides protection from replay and other attacks. Many of the commands and protocols in the specification require a nonce. This structure provides a consistent view of what a nonce is.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_NONCE{
      BYTE nonce[20];
      } TCPA_NONCE;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| BYTE | nonce | This SHALL be the 20 bytes of random data. When created by the TPM the value MUST be the next 20 bytes from the RNG. |

## 4.8 TCPA_AUTHDATA

*Start of informative comment:*

The authorization data is the information that is saved or passed to provide proof of ownership of an entity. For version 1 this area is always 20 bytes.

*End of informative comment.*

**Definition**

typedef BYTE tdTCPA_AUTHDATA[20];

**Parameters**

None.

**Descriptions**

When sending authorization data to the TPM the TPM does not validate the decryption of the data. It is the responsibility of the entity owner to validate that the authorization data was properly received by the TPM. This could be done by immediately attempting to open an authorization session.

The owner of the data can select any value for the data

**Redefinitions**

| Typedef | Name | Description |
|---|---|---|
| TCPA_AUTHDATA | TCPA_SECRET | A secret plaintext value used in the authorization process. |
| TCPA_AUTHDATA | TCPA_ENCAUTH | A ciphertext (encrypted) version of authorization data. The encryption mechanism depends on the context. |

## 4.9 TCPA_KEY_HANDLE_LIST

*Start of informative comment:*

TCPA_KEY_HANDLE_LIST is a structure used to describe the handles of all keys currently loaded in to a TPM. See 8.11.1

*End of informative comment.*

**IDL Definition**

```
typedef struct tdTCPA_KEY_HANDLE_LIST {
      UINT16      loaded;
      [size_is(loaded)] TCPA_KEY_HANDLE   handle[];
} TCPA_KEY_HANDLE_LIST;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| UINT16 | loaded | The number of keys currently loaded in the TPM. |
| UINT32 | handle | An array of handles, one for each key currently loaded in the TPM. |

**Description**

The order in which keys are reported is manufacturer-specific.

## 4.10 TCPA_KEY_USAGE values

*Start of informative comment:*

This table defines the types of keys that are possible.

Each key has a setting defining the encryption and signature scheme to use. The selection of a key usage value limits the choices of encryption and signature schemes.

*End of informative comment*

| Name | Value | Description |
|---|---|---|
| TPM_KEY_SIGNING | 0x0010 | This SHALL indicate a signing key. The [private] key SHALL be used for signing operations, only. This means that it MUST be a leaf of the Protected Storage key hierarchy. |
| TPM_KEY_STORAGE | 0x0011 | This SHALL indicate a storage key. The key SHALL be used to wrap and unwrap other keys in the Protected Storage hierarchy, only. |
| TPM_KEY_IDENTITY | 0x0012 | This SHALL indicate an identity key. The key SHALL be used for operations that require a TPM identity, only. |
| TPM_KEY_AUTHCHANGE | 0X0013 | This SHALL indicate an ephemeral key that is in use during the ChangeAuthAsym process, only. |
| TPM_KEY_BIND | 0x0014 | This SHALL indicate a key that can be used for TPM_Bind and TPM_Unbind operations only. |
| TPM_KEY_LEGACY | 0x0015 | This SHALL indicate a key that can perform signing and binding operations. The key MAY be used for both signing and binding operations. The TPM_KEY_LEGACY key type is to allow for use by applications where both signing and encryption operations occur with the same key. The use of this key type is deprecated. |

### 4.10.1 Mandatory Key Usage Schemes

*Start of Informative Comment:*

For a given key usage type there are subset of valid encryption and signature schemes.

*End of informative comment*

The key usage value for a key determines the encryption and / or signature schemes which MUST be used with that key. The table below maps the schemes defined by this specification to the defined key usage values. See sections 8.4 and 8.5.

| Name | Allowed Encryption schemes | Allowed Signature Schemes |
|---|---|---|
| TPM_KEY_SIGNING | TCPA_ES_NONE | TCPA_SS_RSASSAPKCS1v15_SHA1<br>TCPA_SS_RSASSAPCKS1V15_DER |
| TPM_KEY_STORAGE | TCPA_ES_RSAESOAEP_SHA1_MGF1 | TCPA_SS_NONE |
| TPM_KEY_IDENTITY | TCPA_ES_NONE | TCPA_SS_RSASSAPKCS1v15_SHA1 |
| TPM_KEY_AUTHCHANGE | TCPA_ES_RSAESOAEP_SHA1_MGF1 | TCPA_SS_NONE |
| TPM_KEY_BIND | TCPA_ES_RSAESOAEP_SHA1_MGF1<br>TCPA_ES_RSAESPKCSV15 | TCPA_SS_NONE |
| TPM_KEY_LEGACY | TCPA_ES_RSAESOAEP_SHA1_MGF1<br>TCPA_ES_RSAESPKCSV15 | TCPA_SS_RSASSAPKCS1v15_SHA1<br>TCPA_SS_RSASSAPKCS1V15_DER |

Where manufacturer specific schemes are used, the strength must be at least that listed in the above table for TPM_KEY_STORAGE, TPM_KEY_IDENTITY and TPM_KEY_AUTHCHANGE key types.

## 4.11 TCPA_AUTH_DATA_USAGE values

*Start of informative comment:*

The indication to the TPM when authorization sessions for an entity are required. The only two options at this time are always or never. Future versions may allow for more complex decisions regarding authorization checking.

*End of informative comment.*

| Name | Value | Description |
|---|---|---|
| TPM_AUTH_NEVER | 0x00 | This SHALL indicate that usage of the key without authorization is permitted. |
| TPM_AUTH_ALWAYS | 0x01 | This SHALL indicate that on each usage of the key the authorization MUST be performed. |
| | | All other values are reserved for future use. |

## 4.12 TCPA_KEY_FLAGS

*Start of informative comment*

This table defines the meanings of the bits in a TCPA_KEY_FLAGS structure used in TCPA_STORE_ASYMKEY and TCPA_CERTIFY_INFO.

*End of informative comment*

**TCPA_KEY_FLAGS Values**

| Name | Mask Value | Description |
|---|---|---|
| redirection | 0x00000001 | This mask value SHALL indicate the use of redirected output. |
| migratable | 0x00000002 | This mask value SHALL indicate that the key is migratable. |
| volatileKey | 0x00000004 | This mask value SHALL indicate that the key MUST be unloaded upon execution of the TPM_Init/TPM_Startup sequence. |

The value of TCPA_KEY_FLAGS MUST be decomposed into individual mask values. The presence of a mask value SHALL have the effect described in the above table

## 4.13 Flags and persistent data structures

*Informative comment:*

The TPM maintains flags in volatile and non-volatile areas. These flags indicate the status of TPM enabling, TPM ownership and TPM activation. The TPM also maintains data in volatile and non-volatile areas. Only certain data are required to be stored in non-volatile areas (other data *may* be stored in non-volatile areas, but are not *required* to be stored in non-volatile areas).

The setting of flags requires either authorization by the TPM Owner or the assertion of physical presence at the platform. The nature of assertion of physical presence is a manufacturer option. There are many methods of making the assertion and manufacturers can select any number of options. The underlying theme is that no remote entity should be able to change the status of the TPM without either knowledge of the TPM Ownership authentication or physical presence next to the platform.

One method of providing the physical presence assertion is to have the TPM accept commands during a period when the operation of the platform is constrained. In a PC, the method might operate during the POST and require input from the user via the keyboard. The TPM would allow access to the command until execution of some critical point and the POST process informed the TPM that it should no longer accept the commands.

*End of informative comment.*

### 4.13.1 TCPA persistent data

*Informative comment:*

Purely for the convenience of listing such data together, this structure contains the minimum set of TCPA data that are required to be persistent.

*End of informative comment.*

**IDL Definition**

```
typedef struct tdTCPA_PERSISTENT_DATA{
       BYTE  revMajor;
       BYTE  revMinor;
       TCPA_NONCE  tpmProof;
       TCPA_PUBKEY manuMaintPub;
       TCPA_KEY    endorsementKey;
       TCPA_SECRET ownerAuth;
       TCPA_KEY    srk;
       TCPA_DIRVALUE*    dir;
       BYTE* rngState;
       BYTE ordinalAuditStatus;
}TCPA_PERSISTENT_DATA;
```

**Type**

These data exist in TPM shielded-locations, only, and SHALL be non-volatile. Other TCPA data MAY be persistent, except when specifically prohibited (by an IsVolatile flag, for example).

**Description**

**Types of Persistent Data**

| Type | Name | Description |
|---|---|---|
| BYTE | revMajor | This is the TPM major revision indicator. This SHALL be set by the TPME, only. The default value is manufacturer-specific. |
| BYTE | revMinor | This is the TPM minor revision indicator. This SHALL be set by the TPME, only. The default value is manufacturer-specific. |
| TCPA_NONCE | tpmProof | This is a random number that each TPM maintains to validate blobs in the SEAL and other processes. The default value is manufacturer-specific. |
| TCPA_PUBKEY | manuMaintPub | This is the manufacturer's public key to use in the maintenance operations. The default value is manufacturer-specific. |
| TCPA_KEY | endorsementKey | This is the TPM's endorsement key pair. See 9.2. The default value is manufacturer-specific. |
| TCPA_SECRET | ownerAuth | This is the TPM-Owner's authorization data. See 5.11.1. The default value is manufacturer-specific. |
| TCPA_KEY | srk | This is the TPM's StorageRootKey. See 5.11.1. The default value is manufacturer-specific. |
| TCPA_DIRVALUE* | dir | These are the DataIntegrityRegisters. There MUST be at least one DIR. See, for example, 6.3.4. The default |

| | | |
|---|---|---|
| | | value of a DIR is zero. |
| BYTE* | rngState | State information describing the random number generator. The default state and subsequent states are described in 10.5. |
| BYTE[] | ordinalAuditStatus | Table indicating which ordinals are being audited. See section 8.12 |

### 4.13.2 TCPA_PERSISTENT_FLAGS Structure

*Start of informative comment:*

The persistent flags allow the TPM to maintain internal state across TPM_Init cycles. These flags include flags to indicate activation status and physical presence requirements.

The TPM allows two methods for providing proof of physical presence: hardware and command. The platform manufacturer decides which to provide or allow by setting the values for physicalPresenceHWEnable and physicalPresenceCMDEnable based in the design of the platform and customer requirements. Once set, the manufacturer must lock their states by setting the physicalPresenceLifetimeLock.

The logical ORing of the hardware signal with the PhysiallyPresence flags allows the platform manufacturer to: Allow either method to override the other. Allow one method exclusively. Or disallow both, preventing the local commands from ever executing.

*End of informative comment.*

```
typedef struct tdTCPA_PERSISTENT_FLAGS{
       BOOL disable;
       BOOL ownership;
       BOOL deactivated;
       BOOL readPubek;
       BOOL disableOwnerClear;
       BOOL allowMaintenance;
       BOOL physicalPresenceLifetimeLock;
       BOOL physicalPresenceHWEnable;
       BOOL physicalPresenceCMDEnable;
       BOOL CEKPUsed;
} TCPA_PERSISTENT_FLAGS;
```

**Type**

TPM shielded location: These flags exist only in a TPM shielded-location and SHALL be non-volatile. Other flags MAY be persistent, except when specifically prohibited.

**Parameters**

| Type | Name | Description |
|---|---|---|
| BOOL | disable | The state of the disable flag. See 8.14. The default state is TRUE |
| BOOL | ownership | The ability to install an owner. See 8.12.5. The default state is TRUE. |
| BOOL | deactivated | The state of the inactive flag. See 8.15. The default state is TRUE. |
| BOOL | readPubek | The ability to read the PUBEK without owner authorization. See 9.2.2. The default state is TRUE. |
| BOOL | disableOwnerClear | Whether the owner authorized clear commands are active. See 8.10.6. The default state is FALSE. |
| BOOL | allowMaintenance | Whether the TPM Owner may create a maintenance archive. See 7.3.1. The default state is TRUE. |
| BOOL | physicalPresenceLifetimeLock | This bit can only be set to TRUE; it cannot be set to FALSE except during the manufacturing process.<br>FALSE: The state of either physicalPresenceHWEnable or |

| | | |
|---|---|---|
| | | physicalPresenceCMDEnable MAY be changed. (DEFAULT)<br>**TRUE:** The state of either physicalPresenceHWEnable or physicalPresenceCMDEnable MUST NOT be changed for the life of the TPM. |
| `BOOL` | `physicalPresenceHWEnable` | **FALSE:** Disable the hardware signal indicating physical presence. (DEFAULT)<br>**TRUE:** Enables the hardware signal indicating physical presence. |
| `BOOL` | `physicalPresenceCMDEnable` | **FALSE:** Disable the command indicating physical presence. (DEFAULT)<br>**TRUE:** Enables the command indicating physical presence. |
| `BOOL` | `CEKPUsed` | **TRUE:** The PRIVEK and PUBEK were created using TPM_CreateEndorsementKeyPair.<br>**FALSE:** The PRIVEK and PUBEK were created using a manufacturers process.<br>**NOTE:** This flag has no default value as the key pair MUST be created by one or the other mechanism. |

Description

The data structure TCPA_PERSISTENT_FLAGS SHALL exist in a TPM shielded-location, only, and SHALL be non-volatile.

The physicalPresenceHWEnable and physicalPresenceCMDEnable flags MUST mask their respective signals before further processing. The hardware signal, if enabled by the physicalPresenceHWEnable flag, MUST be logically ORed with the PhysicalPresence flag, if enabled, to obtain the final physical presence value used to allow or disallow local commands.

**Actions**

1. **Disable flag**
   a. If disable has the value of TRUE the following commands will execute with their normal protections
      i. TPM_Reset
      ii. TPM_Init
      iii. TPM_Startup
      iv. TPM_SaveState
      v. TPM_SHA1Start
      vi. TPM_SHA1Update
      vii. TPM_SHA1Complete
      viii. TPM_SHA1CompleteExtend
      ix. TSC_PhysicalPresence
      x. TPM_OIAP
      xi. TPM_OSAP

xii. TPM_GetCapability

xiii. TPM_Extend

xiv. TPM_OwnerSetDisable

xv. TPM_PhysicalEnable

xvi. TPM_ContinueSelfTest

xvii. TPM_SelfTestFull

xviii. TPM_GetTestResult

b. All other commands SHALL return TCPA_DISABLED.

**2. Ownership flag**

a. If ownership has the value of FALSE, then any attempt to install an owner fails with the error value TCPA_INSTALL_DISABLED.

**3. Deactivated flag**

a. This flag sets the state of TCPA_VOLATILE_FLAGS -> deactivated upon initialization.

**4. readPubek**

a. If readPubek is TRUE then the TPM_ReadPubek will return the PUBEK, if FALSE the command will return TCPA_DISABLED_CMD.

**5. DisableOwnerClear**

a. If disableOwnerClear is TRUE then the clear commands requiring owner authorization will return TCPA_CLEAR_DISABLED, if false the commands will execute.

### 4.13.3 TCPA_VOLATILE_FLAGS Structure

*Start of informative comment*

Despite its name, the data structure TCPA_VOLATILE_FLAGS may be stored in non-volatile media. To do so may or may not be advantageous, depending on circumstances. If TCPA_VOLATILE_FLAGS is held in non-volatile storage, the operation of TPM_SaveState is simplified.

TPM_Extend is not permitted to operate when a TPM is deactivated. This is because a deactivated TPM performs no useful service until a platform is rebooted, at which point the PCRs are reset.

TPM_GetCapability and TPM_CreateEndorsementKey may be called before TPM_Startup. This is necessary because TPM_Startup will fail unless an endorsement key exists.

Updating auditDigest is unnecessary when a TPM is deactivated. This is because a deactivated TPM performs no useful service until a platform is rebooted, at which point the auditDigest is reset.

*End of informative comment*

**IDL Definition**

```
typedef struct tdTCPA_VOLATILE_FLAGS{
       BOOL deactivated;
       BOOL disableForceClear;
       BOOL physicalPresence;
       BOOL physicalPresenceLock;
       BOOL postInitialise;
} TCPA_VOLATILE_FLAGS;
```

**Type**

TPM shielded location

**Parameters**

| Type | Name | Description |
|---|---|---|
| BOOL | deactivated | Prevents the operation of most capabilities. There is no default state. It is initialized by TPM_Startup to the same value as TCPA_PERSISTENT_FLAGS -> deactivated. TPM_SetTempDeactivated sets it to TRUE. |
| BOOL | disableForceClear | Prevents the operation of TPM_ForceClear when TRUE. The default state is FALSE. TPM_DisableForceClear sets it to TRUE. |
| BOOL | physicalPresence | Indicates that a User is physically present when TRUE. The default state is FALSE (User is not physically present) |
| BOOL | physicalPresenceLock | Indicates whether changes to the physicalPresence flag are permitted. TPM_Startup/ST_CLEAR sets PhysicalPresence to its default state of FALSE (allow changes to PhysicalPresence flag). The meaning of TRUE is: Do not allow further changes to PhysicalPresence flag. TSC_PhysicalPresence can change the state of physicalPresenceLock. |
| BOOL | postInitialise | Prevents the operation of most capabilities. There is no default state. It is initialized by TPM_Init to TRUE. |

| | | |
|---|---|---|
| | | TPM_Startup sets it to FALSE. |

**Description**

The data structure TCPA_VOLATILE_FLAGS SHALL exist only in a TPM shielded-location.

The data structure TCPA_VOLATILE_FLAGS MAY be held in non-volatile storage.

**Actions**

1. **Deactivated flag**
   a. If deactivated is TRUE the following commands SHALL execute with their normal protections
      i. TPM_Reset
      ii. TPM_Init
      iii. TPM_Startup
      iv. TPM_SaveState
      v. TPM_SHA1Start
      vi. TPM_SHA1Update
      vii. TPM_SHA1Complete
      viii. TPM_SHA1CompleteExtend
      ix. TSC_PhysicalPresence
      x. TPM_OIAP
      xi. TPM_OSAP
      xii. TPM_GetCapability
      xiii. TPM_TakeOwnership
      xiv. TPM_OwnerSetDisable
      xv. TPM_PhysicalDisable
      xvi. TPM_PhysicalEnable
      xvii. TPM_PhysicalSetDeactivated
      xviii. TPM_ContinueSelfTest
      xix. TPM_SelfTestFull
      xx. TPM_GetTestResult
   b. All other commands SHALL return TCPA_DEACTIVATED.
2. **DisableForceClear**

   If disableForceClear is TRUE then the TPM_ForceClear command returns TCPA_CLEAR_DISABLED, if FALSE then the command will execute.
3. **PhysicalPresence**

   If physicalPresence is TRUE and TCPA_PERSISTENT_FLAGS -> physicalPresenceCMDEnable is TRUE, the TPM MAY assume that the Owner is physically present. If physicalPresence is FALSE, the TPM MUST assume that the Owner is physically absent. Note that this physicalPresence is exclusive of the unambiguous physical presence indication required for TPM_PhysicalEnable. They MAY be the same hardware signal depending on the design of the platform and TPM.

4. **physicalPresenceLock**

   If physicalPresenceLock is TRUE, TSC_PhysicalPresence MUST NOT change the physicalPresence flag. If physicalPresenceLock is FALSE, TSC_PhysicalPresence will operate.

5. **postInitialise**

   a. If postInitialise is TRUE the following commands SHALL execute with their normal protections:

      i. TPM_Startup

      ii. TPM_CreateEndorsementKey

      iii. TPM_GetCapability

      iv. TPM_ContinueSelfTest

      v. TPM_SelfTestFull

      vi. TPM_GetTestResult

   b. All other commands SHALL set the flag TCPA_VOLATILE_FLAGS -> postInitialise to FALSE, set TCPA_VOLATILE_FLAGS -> deactivated to TRUE, and return TCPA_INVALID_POSTINIT

## 4.14 TCPA_PAYLOAD_TYPE

*Start of informative comment:*

This structure specifies the type of payload in various messages.

*End of informative comment.*

**Definition**

```
typedef unsigned char TCPA_PAYLOAD_TYPE;
```

**TCPA_PAYLOAD_TYPE Values**

| Value | Name | Comments |
|---|---|---|
| 0x01 | TCPA_PT_ASYM | The entity is an asymmetric key |
| 0x02 | TCPA_PT_BIND | The entity is bound data |
| 0x03 | TCPA_PT_MIGRATE | The entity is a migration blob |
| 0x04 | TCPA_PT_MAINT | The entity is a maintenance blob |
| 0x05 | TCPA_PT_SEAL | The entity is sealed data |
| 0x06 – 0x7F | | Reserved for future use by TCPA |
| 0x80 – 0xFF | | Vendor specific payloads |

## 4.15 TCPA_ENTITY_TYPE

*Start of informative comment*

This specifies the types of entity that are supported by the TPM

*End of informative comment*

**TCPA_ENTITY_TYPE Values**

| Value | Event Name | Comments |
|---|---|---|
| 0x0001 | TCPA_ET_KEYHANDLE | The entity is a keyHandle |
| 0x0002 | TCPA_ET_OWNER | The entity is the TPM Owner |
| 0x0003 | TCPA_ET_DATA | The entity is some data |
| 0x0004 | TCPA_ET_SRK | The entity is the SRK |
| 0x0005 | TCPA_ET_KEY | The entity is a key |

**Description**

For the entity type of TCPA_ET_OWNER the associated key handle MUST be 0x40000001

For the entity type of TCPA_ET_SRK the associated key handle MUST be 0x40000000

## 4.16 TCPA_STARTUP_TYPE

*Start of informative comment:*

To specify what type of startup is occurring

*End of informative comment.*

**TCPA_STARTUP_TYPE Values**

| Value | Event Name | Comments |
|---|---|---|
| 0x0001 | TCPA_ST_CLEAR | The TPM is starting up from a clean state |
| 0x0002 | TCPA_ST_STATE | The TPM is starting up from a saved state |
| 0x0003 | TCPA_ST_DEACTIVATED | The TPM is to startup and set the deactivated flag to TRUE |

## 4.17 TCPA_PROTOCOL_ID

*Start of informative comment:*

This value identifies the protocol in use.

*End of informative comment.*

**Definition**

```
typedef UINT16 TCPA_PROTOCOL_ID;
```

**TCPA_PROTOCOL_ID Values**

| Value | Event Name | Comments |
|---|---|---|
| 0x0001 | TCPA_PID_OIAP | The OIAP protocol. See 5.2.1 |
| 0x0002 | TCPA_PID_OSAP | The OSAP protocol. See 5.2.4 |
| 0x0003 | TCPA_PID_ADIP | The ADIP protocol. See 5.4 |
| 0X0004 | TCPA_PID_ADCP | The ADCP protocol. See 5.6 |
| 0X0005 | TCPA_PID_OWNER | The protocol for taking ownership of a TPM. See 5.11 |

## 4.18 TCPA_ALGORITHM_ID

*Start of informative comment*

This table defines the types of algorithms which may be supported by the TPM.

*End of informative comment*

**Definition**

**TCPA_ALGORITHM_ID values**

| Name | Value | Description |
|---|---|---|
| TCPA_ALG_RSA | 0x00000001 | The RSA algorithm. |
| TCPA_ALG_DES | 0x00000002 | The DES algorithm |
| TCPA_ALG_3DES | 0X00000003 | The 3DES algorithm |
| TCPA_ALG_SHA | 0x00000004 | The SHA1 algorithm |
| TCPA_ALG_HMAC | 0x00000005 | The RFC 2104 HMAC algorithm |
| TCPA_ALG_AES | 0x00000006 | The AES algorithm |

The TPM MUST support the algorithms TCPA_ALG_RSA, TCPA_ALG_SHA, TCPA_ALG_HMAC.

## 4.19 TCPA_PHYSICAL_PRESENCE

| Name | Value | Description |
|---|---|---|
| TCPA_PHYSICAL_PRESENCE_LIFETIME_LOCK | 0x0080h | Sets the physicalPresenceLifetimeLock to TRUE |
| TCPA_PHYSICAL_PRESENCE_HW_ENABLE | 0x0040h | Sets the physicalPresenceHWEnable to TRUE |
| TCPA_PHYSICAL_PRESENCE_CMD_ENABLE | 0x0020h | Sets the physicalPresenceCMDEnable to TRUE |
| TCPA_PHYSICAL_PRESENCE_NOTPRESENT | 0x0010h | Sets PhysicalPresence = FALSE |
| TCPA_PHYSICAL_PRESENCE_PRESENT | 0x0008h | Sets PhysicalPresence = TRUE |
| TCPA_PHYSICAL_PRESENCE_LOCK | 0x0004h | Sets PhysicalPresenceLock = TRUE |

## 4.20 TCPA_KEY_PARMS

*Start of informative comment:*

This provides a standard mechanism to define the parameters used to generate a key pair, and to store the parts of a key shared between the public and private key parts.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_KEY_PARMS {
      TCPA_ALGORITHM_ID algorithmID;
      TCPA_ENC_SCHEME encScheme;
      TCPA_SIG_SCHEME sigScheme;
      UINT32 parmSize;
      [size_is(parmSize)] BYTE* parms;
} TCPA_KEY_PARMS;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_ALGORITHM_ID | algorithmID | This SHALL be the key algorithm in use |
| UINT32 | parmSize | This SHALL be the size of the parms field in bytes |
| TCPA_ENC_SCHEME | encScheme | This SHALL be the encryption scheme that the key uses to encrypt information see section 8.4 |
| TCPA_SIG_SCHEME | sigScheme | This SHALL be the signature scheme that the key uses to perform digital signatures see section 8.5 |
| BYTE[] | parms | This SHALL be the parameter information dependant upon the key algorithm. |

**Descriptions**

The contents of the 'parms' field will vary depending upon algorithmId:

| Algorithm Id | PARMS Contents |
|---|---|
| TCPA_ALG_RSA | A structure of type TCPA_RSA_KEY_PARMS |
| TCPA_ALG_DES | No content |
| TCPA_ALG_3DES | No content – Need description of key size (3 full keys etc) and mode EDE etc. |
| TCPA_ALG_SHA | No content |
| TCPA_ALG_HMAC | No content |
| TCPA_ALG_AES | No content – Need description of key size (128, 192, 256) |

### 4.20.1 TCPA_RSA_KEY_PARMS

*Start of informative comment:*

This structure describes the parameters of an RSA key.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_RSA_KEY_PARMS {
        UINT32 keyLength;
        UINT32 numPrimes;
        UINT32 exponentSize;
        BYTE[] exponent;
} TCPA_RSA_KEY_PARMS;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| UINT32 | keyLength | This specifies the size of the RSA key in bits |
| UINT32 | numPrimes | This specifies the number of prime factors used by this RSA key. |
| UINT32 | exponentSize | This SHALL be the size of the exponent. If the key is using the exponent from 10.4.1 then the exponentSize MUST be 0. |
| BYTE[] | exponent | The public exponent of this key |

## 4.21 TCPA_CHANGEAUTH_VALIDATE

*Start of informative comment:*

This structure provides an area that will stores the new authorization data and the challenger's nonce.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_CHANGEAUTH_VALIDATE {
      TCPA_SECRET newAuthSecret;
      TCPA_NONCE n1;
} TCPA_CHANGEAUTH_VALIDATE;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_SECRET | newAuthSecret | This SHALL be the new authorization data for the target entity |
| TCPA_NONCE | n1 | This SHOULD be a nonce, to enable the caller to verify that the target TPM is on-line. |

## 4.22 TCPA_MIGRATE_SCHEME

*Start of informative comment:*

The scheme indicates how the StartMigrate command should handle the migration of the encrypted blob.

*End of informative comment.*

**Definition**

**TCPA_MIGRATE_SCHEME values**

| Name | Value | Description |
|---|---|---|
| TCPA_MS_MIGRATE | 0x0001 | A public key that can be used with all TCPA migration commands other than 'ReWrap' mode. |
| TCPA_MS_REWRAP | 0x0002 | A public key that can be used for the ReWrap mode of TPM_CreateMigrationBlob. |
| TCPA_MS_MAINT | 0x0003 | A public key that can be used for the Maintenance commands |

## 4.23 TCPA_MIGRATIONKEYAUTH

*Start of informative comment:*

This structure provides the proof that the associated public key has TPM Owner authorization to be a migration key.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_MIGRATIONKEYAUTH{
      TCPA_PUBKEY migrationKey;
      TCPA_MIGRATE_SCHEME migrationScheme;
      TCPA_DIGEST digest;
} TCPA_MIGRATIONKEYAUTH;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_PUBKEY | migrationKey | This SHALL be the public key of the migration facility |
| TCPA_MIGRATE_SCHEME | migrationScheme | This shall be the type of migration operation. |
| TCPA_DIGEST | digest | This SHALL be the digest value of the concatenation of migration key, migration scheme and tpmProof |

## 4.24 TCPA_AUDIT_EVENT structure

*Start of informative comment:*

This structure reports the contents of the audit log. The entries in the log, if hashed together should equal the current hash value held by the TPM. Mismatches indicate attacks on the system or failures to properly audit events.

The 1 version has the minimal information necessary to recreate the history of audited operations.

Future versions may add additional information.

*End of informative comment.*

**IDL Definition**

```
typedef struct tdTCPA_AUDIT_EVENT{
      TCPA_COMMAND_CODE ordinal;
      TCPA_RESULT returncode;
} TCPA_AUDIT_EVENT;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_COMMAND_CODE | ordinal | Ordinal of the command |
| TCPA_RESULT | returncode | Return code for the command |

## 4.25 PCR Structures

*Start of informative comment:*

The PCR structures expose the information in PCR register, allow for selection of PCR register or registers in the SEAL operation and define what information is held in the PCR register.

These structures are in use during the wrapping of keys and sealing of blobs.

*End of informative comment.*

### 4.25.1 TCPA_EVENT_CERT

*Start of informative comment:*

Certificate structure to use when adding EV_CODE_CERT events to the log

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_EVENT_CERT {
      TCPA_DIGEST certificateHash;
      TCPA_DIGEST entityDigest;
      BOOL digestChecked;
      BOOL digestVerified;
      UINT32 issuerSize;
      [size_is (IssuerSize)] BYTE * issuer;
) TCPA_EVENT_CERT;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_DIGEST | certificateHash | Hash of the entire VE certificate |
| TCPA_DIGEST | entityDigest | Actual digest value of the entity |
| BOOL | digestChecked | TRUE if the entity logging this event checked the measured value against the digest value in the certificate.<br>FALSE if no checking was attempted. |
| BOOL | digestVerified | Only valid when DigestChecked is TRUE.<br>TRUE if measured value matches digest value in certificate, FALSE otherwise. |
| UINT32 | issuerSize | Size of the Issuer parameter |
| BYTE* | issuer | Actual issuer certificate |

### 4.25.2 TCPA_PCR_EVENT

*Start of informative comment:*

Individual events are stored in the TCPA_PCR_EVENT variably sized data structure.

TCPA defines the following event/supporting information types:

EventType Values:

| Value | Event Name | Comments |
|---|---|---|
| 0 | EV_CODE_CERT | The TPM_Extend event is in response to loading a firmware or software component for which a VE certificate was available. Event points to the VE certificate that shipped with the platform firmware or software (or discovered by other means). Size indicates the length of this structure. ExtendValue is the digest of the firmware, software or other code loaded. Certificates are much too large to put into the log in the Pre-OS environment. Validation of Certificates is unlikely in the Pre-OS environment. The event MUST point to a TCPA_EVENT_CERT structure. |
| 1 | EV_CODE_NOCERT | The event was in response to loading a firmware or other software component, but no VE certificate was found. The size is 0 and Event is unused. However, ExtendValue is the digest of the firmware discovered. Absence of a VE certificate does not indicate lack of trust; it merely indicates that a VE certificate was not available at this point in boot. Upper-level software may be able to obtain such certificates. |
| 2 | EV_XML_CONFIG | The event describes the platform configuration. The supporting information is a platform or firmware-defined XML data structure that indicates security-relevant hardware configuration information. The event logged to TPM_Extend is the SHA-1 digest of the XML data structure, and the firmware guarantees that the configuration stated in the data structure is in effect when the firmware relinquishes control to the next module in boot. Size is the size in bytes of the XML data structure, and Event points to the data structure itself. The information may include size of physical memory, number of processors, chipset configuration, buses discovered, and processor/bus frequencies. Firmware vendors are free to define the XML reporting structure and select those parameters that are important for their platforms. |
| 3 | EV_NO_ACTION | The action was not performed. The corresponding DIGEST structure MUST be 0x1 (a single binary digit in the LSB of the DIGEST structure) and this value MUST also be logged to the TPM using the corresponding TPM_Extend operation. A supporting data structure may be supplied containing information that describes why the event did not occur. If such supporting information is supplied, it should be well-formed XML. However, this supporting information is not required. |
| 4 | EV_SEPARATOR | A list of actions was complete. This event must be used if more than one event can be logged to the TPM and upper level software needs to be informed that logging was completed. |
| 5 | EV_ACTION | A logged event. This is a Unicode string with the content defined by the Platform Specific specifications. |
| 6 | EV_PLATFORM_SPECIFIC | Implementation specification defined data. |

| | | |
|---|---|---|
| 7 — $(2^{16}-1)$ | Reserved | TCPA reserved event types |
| $2^{16}$ — $(2^{32}-1)$ | User-definable | Undefined and free for general-purpose use |

Additional event types may be defined for TCPA usage in specific computing platforms (for example, the PC)

***End of informative comment***

### 4.25.3 TCPA_PCR_SELECTION

*Start of informative comment:*

This structure provides a standard method of specifying a list of PCR registers.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_PCR_SELECTION {
      UINT16 sizeOfSelect;
      [size_is(sizeOfSelect)] BYTE pcrSelect[];
      } TCPA_PCR_SELECTION;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| UINT16 | sizeOfSelect | The size in bytes of the pcrSelect structure |
| BYTE | pcrSelect | This SHALL be a bit map that indicates if a PCR is active or not |

**Description**

When the least-significant-bit of byte [N+1] of pcrSelect is butted against the most-significant-bit of byte [N] of pcrSelect for (15>=N>=0), the contiguous bit array so formed SHALL represent PCR indices in monotonically increasing order, starting from PCR index zero represented by bit 0 of byte 0 of pcrSelect.

The state of each bit in pcrSelect indicates whether a PCR register is selected or not. When the bit is 1 then the corresponding PCR is selected, if 0 the PCR is not selected.

The TPM MUST support a minimum sizeOfSelect of 2, larger sizes are allowable. The TPM MAY support TCPA_PCR_SELECTION structures with a larger size.

### 4.25.4 TCPA_PCR_COMPOSITE

*Start of informative comment:*

The composite structure provides the index and value of the PCR register to be used when creating the value that SEALS an entity to the composite.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_PCR_COMPOSITE {
      TCPA_PCR_SELECTION select;
      UINT32 valueSize;
      [size_is(valueSize)] TCPA_PCRVALUE pcrValue[];
   } TCPA_PCR_COMPOSITE;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_PCR_SELECTION | select | This SHALL be the indication of which PCR values are active |
| UINT32 | valueSize | This SHALL be the size of the pcrValue field |
| TCPA_PCRVALUE | pcrValue[] | This SHALL be an array of TCPA_PCRVALUE structures. The values come in the order specified by the select parameter and are concatenated into a single blob |

### 4.25.5 TCPA_PCR_INFO

*Start of informative comment*

The TCPA_PCR_INFO structure contains the information related to the wrapping of a key or the sealing of data, to a set of PCRs.

*End of informative comment*

**Definition**

```
typedef struct tdTCPA_PCR_INFO{
      TCPA_PCR_SELECTION pcrSelection;
      TCPA_COMPOSITE_HASH digestAtRelease;
      TCPA_COMPOSITE_HASH digestAtCreation;
      } TCPA_PCR_INFO;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_PCR_SELECTION | pcrSelection | This SHALL be the selection of PCRs to which the data or key is bound. |
| TCPA_COMPOSITE_HASH | digestAtRelease | This SHALL be the digest of the PCR indices and PCR values to verify when revealing Sealed Data or using a key that was wrapped to PCRs. |
| TCPA_COMPOSITE_HASH | digestAtCreation | This SHALL be the composite digest value of the PCR values, at the time when the sealing is performed. |

## 4.26 Storage Structures

### 4.26.1 TCPA_STORED_DATA

*Start of informative comment:*

The definition of this structure is necessary to ensure the enforcement of security properties.

This structure is in use by the TPM_Seal and TPM_Unseal commands to identify the PCR index and values that must be present to properly unseal the data.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_STORED_DATA {
       TCPA_VERSION ver;
       UINT32 sealInfoSize;
       [size_is(sealInfoSize)] BYTE* sealInfo;
       UINT32 encDataSize;
       [size_is(encDataSize)] BYTE* encData;
       } TCPA_STORED_DATA;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | ver | Version number defined in section 4.5. |
| UINT32 | sealInfoSize | Size of the sealInfo parameter |
| BYTE* | sealInfo | This SHALL be a structure of type TCPA_PCR_INFO or a 0 length array if the data is not bound to PCRs. |
| UINT32 | encDataSize | This SHALL be the size of the encData parameter |
| BYTE* | encData | This shall be an encrypted TCPA_SEALED_DATA structure containing the confidential part of the data. |

**Descriptions**

This structure is created during the TPM_Seal process. The confidential data is encrypted using a non-migratable key. When the TPM_Unseal decrypts this structure the TPM_Unseal uses the public information in the structure to validate the current configuration and release the decrypted data.

### 4.26.2 TCPA_SEALED_DATA

*Start of informative comment:*

This structure contains confidential information related to sealed data, including the data itself.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_SEALED_DATA {
      TCPA_PAYLOAD_TYPE payload;
      TCPA_SECRET authData;
      TCPA_NONCE tpmProof;
      TCPA_DIGEST storedDigest;
      UINT32 dataSize;
      [size_is(dataSize)] BYTE* data;
      } TCPA_SEALED_DATA;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_PAYLOAD_TYPE | payload | This SHALL indicate the payload type of TCPA_PT_SEAL |
| TCPA_SECRET | authData | This SHALL be the authorization data for this value |
| TCPA_NONCE | tpmProof | This SHALL be a copy of TPM_PERSISTENT_FLAGS -> tpmProof |
| TCPA_DIGEST | storedDigest | This SHALL be a digest of the TCPA_STORED_DATA structure, excluding the fields TCPA_STORED_DATA -> encDataSize and TCPA_STORED_DATA -> encData. |
| UINT32 | dataSize | This SHALL be the size of the data parameter |
| BYTE* | data | This SHALL be the data to be sealed |

**Description**

To tie the TCPA_STORED_DATA structure to the TCPA_SEALED_DATA structure this structure contains a digest of the containing TCPA_STORED_DATA structure.

The digest calculation does not include the encDataSize and encData parameters.

### 4.26.3 TCPA_SYMMETRIC_KEY

*Start of informative comment:*

This structure describes a symmetric key, used during the process 0 "Collating a Request for a Trusted Platform Module Identity".

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_SYMMETRIC_KEY {
      TCPA_ALGORITHM_ID algId;
      TCPA_ENC_SCHEME encScheme;
      UINT16 size;
      [size_is(size)] BYTE* data;
} TCPA_SYMMETRIC_KEY;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_ALGORITHM_ID | algId | This SHALL be the algorithm identifier of the symmetric key. |
| TCPA_ENC_SCHEME | encScheme | This SHALL fully identify the manner in which the key will be used for encryption operations. |
| UINT16 | size | This SHALL be the size of the data parameter in bytes |
| BYTE* | data | This SHALL be the symmetric key data |

### 4.26.4 TCPA_BOUND_DATA

*Start of informative comment:*

This structure is defined because it is used by a TPM_UnBind command in a consistency check.

The intent of TCPA is to promote "best practice" heuristics for the use of keys: a signing key shouldn't be used for storage, and so on. These heuristics are used because of the potential threats that arise when the same key is used in different ways. The heuristics minimize the number of ways in which a given key can be used.

One such heuristic is that a key of type TPM_KEY_BIND, and no other type of key, should always be used to create the blob that is unwrapped by TPM_UnBind. Binding is not a TPM function, so the only choice is to perform a check for the correct payload type when a blob is unwrapped by a key of type TPM_KEY_BIND. This requires the blob to have internal structure.

Even though payloadData has variable size, TCPA_BOUND_DATA deliberately does not include the size of payloadData. This is to maximise the size of payloadData that can be encrypted when TCPA_BOUND_DATA is encrypted in a single block. When using TPM_UnBind to obtain payloadData, the size of payloadData is deduced as a natural result of the (RSA) decryption process.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_BOUND_DATA {
      TCPA_VERSION ver;
      TCPA_PAYLOAD_TYPE payload;
      BYTE[] payloadData;
      } TCPA_BOUND_DATA;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | ver | Version number defined in section 4.5. |
| TCPA_PAYLOAD_TYPE | payload | This SHALL be the value TCPA_PT_BIND |
| BYTE[] | payloadData | The bound data |

**Descriptions**

This structure MUST be used for creating data when (wrapping with a key of type TPM_KEY_BIND) or (wrapping using the encryption algorithm TCPA_ES_RSAESOAEP_SHA1_M). If it is not, the TPM_UnBind command will fail.

## 4.27 TCPA_KEY complex

*Start of informative comment:*

The TPA_KEY complex is where all of the information regarding keys is kept. These structures combine to fully define and protect the information regarding an asymmetric key.

This version of the specification only fully defines RSA keys, however the design is such that in the future when other asymmetric algorithms are available the general structure will not change.

One overriding design goal is for a 2048 bit RSA key to be able to properly protect another 2048 bit RSA key. This stems from the fact that the SRK is a 2048 bit key and all identities are 2048 bit keys. A goal is to have these keys only require one decryption when loading an identity into the TPM. The structures as defined meet this goal.

Every TCPA_KEY is allowed only one encryption scheme or one signature scheme (or one of each in the case of legacy keys) throughout its lifetime. Note however that more than one scheme could be used with externally generated keys, by introducing the same key in multiple blobs.

*End of informative comment.*

### 4.27.1 TCPA_KEY

*Start of informative comment:*

The TCPA_KEY structure provides a mechanism to transport the entire asymmetric key pair. The private portion of the key is always encrypted.

The reason for using a size and pointer for the PCR info structure is save space when the key is not bound to a PCR. The only time the information for the PCR is kept with the key is when the key needs PCR info.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_KEY{
      TCPA_VERSION ver;
      TCPA_KEY_USAGE keyUsage;
      TCPA_KEY_FLAGS keyFlags;
      TCPA_AUTH_DATA_USAGE authDataUsage;
      TCPA_KEY_PARMS algorithmParms;
      UINT32 PCRInfoSize;
      BYTE* PCRInfo;
      TCPA_STORE_PUBKEY pubKey;
      UINT32 encSize;
      [size_is(encData)] BYTE* encData;
      } TCPA_KEY;
```

Parameters

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | ver | Version number defined in section 4.5. |
| TCPA_KEY_USAGE | keyUsage | This SHALL be the TCPA key usage that determines the operations permitted with this key |
| TCPA_KEY_FLAGS | keyFlags | This SHALL be the indication of migration, redirection etc. |
| TCPA_AUTH_DATA_USAGE | authDataUsage | This SHALL Indicate the conditions where it is required that authorization be presented. |
| TCPA_KEY_PARMS | algorithmParms | This SHALL be the information regarding the algorithm for this key |
| UINT32 | PCRInfoSize | This SHALL be the length of the pcrInfo parameter. If the key is not bound to a PCR this value SHOULD be 0. |
| BYTE* | PCRInfo | This SHALL be a structure of type TCPA_PCR_INFO, or an empty array if the key is not bound to PCRs. |
| TCPA_STORE_PUBKEY | pubKey | This SHALL be the public portion of the key |
| UINT32 | encSize | This SHALL be the size of the encData parameter. |
| BYTE* | encData | This SHALL be an encrypted<br>TCPA_STORE_ASYMKEY structure<br>TCPA_MIGRATE_ASYMKEY structure |

### 4.27.2 TCPA_STORE_PUBKEY

*Start of informative comment*

This structure can be used in conjunction with a corresponding TCPA_KEY_PARMS to construct a public key which can be unambiguously used.

*End of informative comment*

```
typedef struct tdTCPA_STORE_PUBKEY {
      UINT32 keyLength;
      BYTE[] key;
} TCPA_STORE_PUBKEY;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| UINT32 | keyLength | This SHALL be the length of the key field. |
| BYTE[] | key | This SHALL be a structure interpreted according to the algorithm Id in the corresponding TCPA_KEY_PARMS structure. |

**Descriptions**

The contents of the 'key' field will vary depending upon the corresponding key algorithm:

| Algorithm Id | 'Key' Contents |
|---|---|
| TCPA_ALG_RSA | The RSA public modulus |

### 4.27.3 TCPA_PUBKEY

*Start of informative comment*

The TCPA_PUBKEY structure contains the public portion of an asymmetric key pair. It contains all the information necessary for it's unambiguous usage. It is possible to construct this structure from a TCPA_KEY, using the algorithmParms and pubKey fields.

*End of informative comment*

**Definition**

```
typedef struct tdTCPA_PUBKEY{
      TCPA_KEY_PARMS algorithmParms;
      TCPA_STORE_PUBKEY pubKey;
      } TCPA_PUBKEY;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_KEY_PARMS | algorithmParms | This SHALL be the information regarding this key |
| TCPA_STORE_PUBKEY | pubKey | This SHALL be the public key information |

**Descriptions**

The pubKey member of this structure shall contain the public key for a specific algorithm.

### 4.27.4 TCPA_STORE_ASYMKEY

*Start of informative comment:*

The TCPA_STORE_ASYMKEY structure provides the area to identify the confidential information related to a key. This will include the private key factors for an asymmetric key.

The structure is designed so that encryption of a TCPA_STORE_ASYMKEY structure containing a 2048 bit RSA key can be done in one operation if the encrypting key is 2048 bits.

Using typical RSA notation the structure would include P, and when loading the key include the unencrypted P*Q which would be used to recover the Q value.

To accommodate the future use of multiple prime RSA keys the specification of additional prime factors is an optional capability.

This structure provides the basis of defining the protection of the private key. For the complete description of the entire encryption process see 8.4.1

Changes in this structure MUST be reflected in the TCPA_MIGRATE_ASYMKEY structure (section 4.27.6).

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_STORE_ASYMKEY {             // pos    len       total
      TCPA_PAYLOAD_TYPE payload;                  //   0      1           1
      TCPA_SECRET usageAuth;                      //   1     20          21
      TCPA_SECRET migrationAuth;                  //  21     20          41
      TCPA_DIGEST pubDataDigest;                  //  41     20          61
      TCPA_STORE_PRIVKEY privKey;                 //  61    132-151  193-214
} TCPA_STORE_ASYMKEY;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_PAYLOAD_TYPE | payload | This SHALL set to TCPA_PT_ASYM to indicate an asymmetric key. |
| TCPA_SECRET | usageAuth | This SHALL be the authorization data necessary to authorize the use of this value |
| TCPA_SECRET | migrationAuth | This SHALL be the migration authorization data for a migratable key, or the TPM secret value tpmProof for a non-migratable key created by the TPM.<br><br>If the TPM sets this parameter to the value tpmProof, then the TCPA_KEY.keyFlags.migratable of the corresponding TCPA_KEY structure MUST be set to 0.<br><br>If this parameter is set to the migration authorization data for the key in parameter PrivKey, then the TCPA_KEY.keyFlags.migratable of the corresponding TCPA_KEY structure SHOULD be set to 1. |
| TCPA_DIGEST | pubDataDigest | This SHALL be the digest of the corresponding TCPA_KEY structure, excluding the fields TCPA_KEY.encSize and TCPA_KEY.encData.<br><br>When TCPA_KEY -> pcrInfoSize is 0 then the digest calculation has no input from the pcrInfo field. The pcrInfoSize |

| | | |
|---|---|---|
| | | field MUST always be part of the digest calcuation. |
| TCPA_STORE_PRIVKEY | privKey | This SHALL be the private key data. The privKey can be a variable length which allows for differences in the key format. The maximum size of the area would be 151 bytes. |

### 4.27.5 TCPA_STORE_PRIVKEY

*Start of informative comment:*

This structure can be used in conjunction with a corresponding TCPA_PUBKEY to construct a private key which can be unambiguously used.

*End of informative comment.*

```
typedef struct tdTCPA_STORE_PRIVKEY {
      UINT32 keyLength;
      [size_is(keyLength)] BYTE* key;
} TCPA_STORE_PRIVKEY;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| UINT32 | keyLength | This SHALL be the length of the key field. |
| BYTE* | key | This SHALL be a structure interpreted according to the algorithm Id in the corresponding TCPA_KEY structure. |

**Descriptions**

All migratable keys MUST be RSA keys with two (2) prime factors.

For non-migratable keys, the size, format and contents of privKey.key MAY be vendor specific and MAY not be the same as that used for migratable keys. The level of cryptographic protection MUST be at least as strong as a migratable key.

| Algorithm Id | key Contents |
|---|---|
| TCPA_ALG_RSA | When the numPrimes defined in the corresponding TCPA_RSA_KEY_PARMS field is 2, this shall be one of the prime factors of the key. Upon loading of the key the TPM calculates the other prime factor by dividing the modulus, stated in section 10.4.1: TCPA_RSA_PUBKEY, by this value.<br>The TPM MAY support RSA keys with more than two prime factors. Definition of the storage structure for these keys is left to the TPM Manufacturer. |

### 4.27.6 TCPA_MIGRATE_ASYMKEY

*Start of informative comment:*

The TCPA_MIGRATE_ASYMKEY structure provides the area to identify the private key factors of a asymmetric key while the key is migrating between TPM's.

This structure provides the basis of defining the protection of the private key. For the complete description of the entire encryption process see 7.2.11

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_MIGRATE_ASYMKEY {              // pos    len       total
      TCPA_PAYLOAD_TYPE payload;                     //   0      1           1
      TCPA_SECRET usageAuth;                         //   1     20          21
      TCPA_DIGEST pubDataDigest;                     //  21     20          41
      UINT32 partPrivKeyLen;                         //  41      4          45
      TCPA_STORE_PRIVKEY partPrivKey;                //  45   112-127   157-172
} TCPA_MIGRATE_ASYMKEY;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_PAYLOAD_TYPE | payload | This SHALL set to TCPA_PT_MIGRATE to indicate an migrating asymmetric key or TCPA_PT_MAINT to indicate a maintenance key. |
| TCPA_SECRET | usageAuth | This SHALL be a copy of the usageAuth from the TCPA_STORE_ASYMKEY structure. |
| TCPA_DIGEST | pubDataDigest | This SHALL be a copy of the pubDataDigest from the TCPA_STORE_ASYMKEY structure. |
| UINT32 | partPrivKeyLen | This SHALL be the size of the partPrivKey field |
| TCPA_STORE_PRIVKEY | partPrivKey | This SHALL be the k2 area as defined in section 7.2.11 |

## 4.28 TCPA_CERTIFY_INFO Structure

*Start of informative comment:*

When the TPM certifies a key, it must provide a signature with a TPM identity key on information that describes that key. This structure provides the mechanism to do so.

*End of informative comment.*

**IDL Definition**

```
typedef struct tdTCPA_CERTIFY_INFO{
      TCPA_VERSION version;
      TCPA_KEY_USAGE keyUsage;
      TCPA_KEY_FLAGS keyFlags;
      TCPA_AUTH_DATA_USAGE authDataUsage;
      TCPA_KEY_PARMS algorithmParms;
      TCPA_DIGEST pubkeyDigest;
      TCPA_NONCE data;
      BOOL parentPCRStatus;
      UINT32 PCRInfoSize;
      [size_is(pcrInfoSize)] BYTE* PCRInfo;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | version | TCPA version structure; section 4.5 . |
| TCPA_KEY_USAGE | keyUsage | This SHALL be the same value that would be set in a TCPA_KEY representation of the key to be certified |
| TCPA_KEY_FLAGS | keyFlags | This SHALL be set to the same value as the corresponding parameter in the TCPA_KEY structure that describes the public key that is being certified |
| TCPA_AUTH_DATA_USAGE | authDataUsage | This SHALL be the same value that would be set in a TCPA_KEY representation of the key to be certified |
| TCPA_KEY_PARMS | algorithmParms | This SHALL be the same value that would be set in a TCPA_KEY representation of the key to be certified |
| TCPA_DIGEST | pubKeyDigest | This SHALL be a digest of the value TCPA_KEY -> pubKey -> key in a TCPA_KEY representation of the key to be certified |
| TCPA_NONCE | data | This SHALL be externally provided data. |
| BOOL | parentPCRStatus | This SHALL indicate if any parent key was wrapped to a PCR |
| UINT32 | PCRInfoSize | This SHALL be the size of the pcrInfo parameter. A value of zero indicates that the key is not wrapped to a PCR |
| BYTE* | PCRInfo | This SHALL be the TCPA_PCR_INFO structure. |

## 4.29 TCPA_QUOTE_INFO Structure

*Start of informative comment:*

This structure provides the mechanism for the TPM to quote the current values of a list of PCRs.

*End of informative comment.*

**IDL Definition**

```
typedef struct tdTCPA_QUOTE_INFO{
      TCPA_VERSION version;
      BYTE fixed[4];
      TCPA_COMPOSITE_HASH digestValue;
      TCPA_NONCE externalData,
} TCPA_QUOTE_INFO;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | version | TCPA version structure; section 4.5 |
| BYTE | fixed | This SHALL always be the string 'QUOT' |
| TCPA_COMPOSITE_HASH | digestValue | This SHALL be the result of the composite hash algorithm using the current values of the requested PCR indices. |
| TCPA_NONCE | externalData | 160 bits of externally supplied data |

## 4.30 Identity Structures

### 4.30.1 TCPA_IDENTITY_CONTENTS

*Start of informative comment:*

TPM_MakeIdentity uses this structure and the signature of this structure goes to a privacy CA during the certification process.

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_IDENTITY_CONTENTS {
      TCPA_VERSION              ver
      UINT32                    ordinal,
      TCPA_CHOSENID_HASH        labelPrivCADigest,
      TCPA_PUBKEY               identityPubKey;
} TCPA_IDENTITY_CONTENTS;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | ver | This SHALL be the version specified in section 4.5. |
| UINT32 | ordinal | This SHALL be the ordinal of the TPM_MakeIdentity command. |
| TCPA_CHOSENID_HASH | labelPrivCADigest | This SHALL be the result of hashing the chosen identityLabel and privacyCA for the new TPM identity (see 10.4.6 for details) |
| TCPA_PUBKEY | identityPubKey | This SHALL be the public key structure of the identity key |

### 4.30.2 TCPA_IDENTITY_REQ

*Start of informative comment:*

This structure is sent by the TSS to the Privacy CA to create the identity credential.

*End of informative comment.*

**Parameters**

| Type | Name | Description |
|---|---|---|
| UINT32 | asymSize | This SHALL be the size of the asymmetric encrypted area created by TSS_CollateIdentityRequest |
| UINT32 | symSize | This SHALL be the size of the symmetric encrypted area created by TSS_CollateIdentityRequest |
| TCPA_KEY_PARMS | asymAlgorithm | This SHALL be the parameters for the asymmetric algorithm used to create the asymBlob |
| TCPA_KEY_PARMS | symAlgorithm | This SHALL be the parameters for the symmetric algorithm used to create the symBlob |
| BYTE* | asymBlob | This SHALL be the asymmetric encrypted area from TSS_CollateIdentityRequest |
| BYTE* | symBlob | This SHALL be the symmetric encrypted area from TSS_CollateIdentityRequest |

### 4.30.3 TCPA_IDENTITY_PROOF

Start of informative comment:

This structure is used during the process 0 Collating a Request for a Trusted Platform Module Identity.

End of informative comment.

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | ver | This SHALL be the version specified in section 4.5. |
| UINT32 | labelSize | This SHALL be the size of the label area |
| UINT32 | identityBindingSize | This SHALL be the size of the identitybinding area |
| UINT32 | endorsementSize | This SHALL be the size of the endorsement credential |
| UINT32 | platformSize | This SHALL be the size of the platform credential |
| UINT32 | conformanceSize | This SHALL be the size of the conformance credential |
| TCPA_PUBKEY | identityKey | This SHALL be the public key of the new identity |
| BYTE* | labelArea | This SHALL be the text label for the new identity |
| BYTE* | identityBinding | This SHALL be the signature value of TCPA_IDENTITY_CONTENTS structure from the TPM_MakeIdentity command |
| BYTE* | endorsementCredential | This SHALL be the TPM endorsement credential |
| BYTE* | platformCredential | This SHALL be the TPM platform credential |
| BYTE* | conformanceCredential | This SHALL be the TPM conformance credential |

### 4.30.4 TCPA_ASYM_CA_CONTENTS

*Start of informative comment:*

This structure contains the symmetric key to encrypt the identity credential

*End of informative comment.*

**Definition**

```
typedef struct tdTCPA_ASYM_CA_CONTENTS{
      TCPA_SYMMETRIC_KEY sessionKey;
      TCPA_DIGEST idDigest;
} TCPA_ASYM_CA_CONTENTS;
```

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_SYMMETRIC_KEY | sessionKey | This SHALL be the session key used by the CA to encrypt the TCPA_IDENTITY_CREDENTIAL |
| TCPA_DIGEST | idDigest | This SHALL be the digest of the TPM identity public key that is being certified by the CA |

### 4.30.5 TCPA_SYM_CA_ATTESTATION

*Start of informative comment:*

This structure returned by the Privacy CA with the encrypted identity credential.

*End of informative comment.*

| Type | Name | Description |
|---|---|---|
| UINT32 | credSize | This SHALL be the size of the credential parameter |
| TCPA_KEY_PARMS | algorithm | This SHALL be the indicator and parameters for the symmetric algorithm |
| BYTE* | credential | This is the result of encrypting TPM_IDENTITY_CREDENTIAL using the session_key and the algorithm indicated "algorithm" |

## 4.31 TCPA_CAPABILITY_AREA

*Start of informative comment:*

To identify a capability to be queried.

*End of informative comment.*

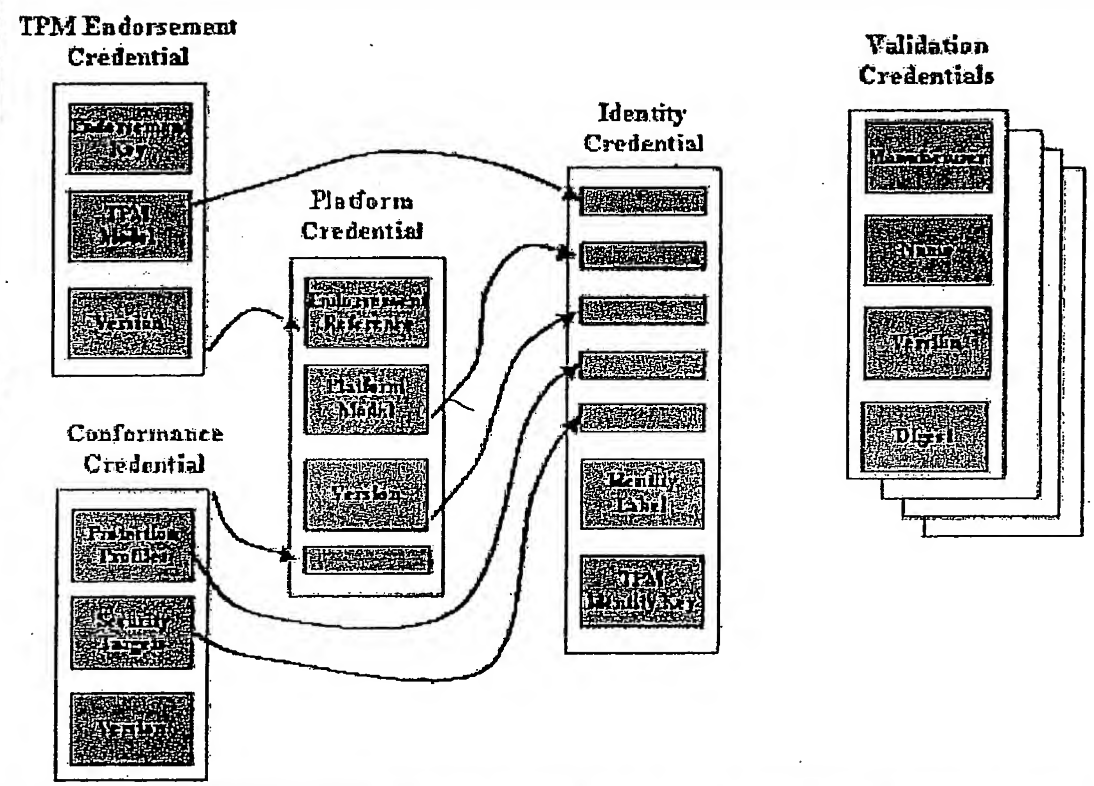
**TCPA_CAPABILITY_AREA Values**

| Value | Capability Name | Comments |
|---|---|---|
| 0x00000001 | TCPA_CAP_ORD | Queries whether a command is supported. |
| 0x00000002 | TCPA_CAP_ALG | Queries whether an algorithm is supported. |
| 0x00000003 | TCPA_CAP_PID | Queries whether a protocol is supported. |
| 0x00000004 | TCPA_CAP_FLAG | Queries whether a flag is on or off. |
| 0x00000005 | TCPA_CAP_PROPERTY | Determines a physical property of the TPM. |
| 0x00000006 | TCPA_CAP_VERSION | Queries the current TPM version. |
| 0x00000007 | TCPA_CAP_KEY_HANDLE | Obtains information about all key handles |
| 0x00000008 | TPM_CAP_CHECK_LOADED | Obtains information about the ability to load a key |
| 0x00000009 | | |
| 0x0000000A | | |
| 0x0000000B | | |

## 4.32 Credentials

*Start of informative comment*

The credentials in use for a TCPA system interlock. The following diagram shows the relationship between the credentials. Credentials, being abstract, are instantiated as tangible, unambiguous entities in Section 9.5 Instantiation of Credentials as Certificates




*End of informative comment*

### 4.32.1 Evidence of Subsystem Endorsement

*Start of informative comment*

The purpose of TPM_ENDORSEMENT_CREDENTIAL is to provide evidence that a TPM correctly implements the protected capabilities and shielded locations of the TCPA specification.

TPM_ENDORSEMENT_CREDENTIAL is an attestation that a genuine TCPA Trusted Platform Module created the PUBEK that is referenced in TPM_ENDORSEMENT_CREDENTIAL. TPM_ENDORSEMENT_CREDENTIAL contains information that a Privacy CA may use in judging whether the Privacy CA will attest to an identity of that TCPA Trusted Platform Module. TPM_ENDORSEMENT_CREDENTIAL contains information that the Privacy CA must use in attesting to an identity of that TCPA Trusted Platform Module.

TPM_ENDORSEMENT_CREDENTIAL is tagged with TCPA_VERSION so as to indicate the version of the capability that created the PUBEK at the time the key was generated. This may be useful in the event that capabilities are field-upgraded.

- PUBEK will be required by the Privacy CA when the Privacy CA attests to a TCPA Trusted Platform Module identity (TPM identity).
- TCPA Trusted Platform Module Endorsement identifies a data structure as TPM_ENDORSEMENT_CREDENTIAL and enables the TPME to sign the data with a key that is not exclusively reserved for signing TPM_ENDORSEMENT_CREDENTIAL.
- tpme_reference is the means of referencing the TPME. may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity, and is required by the Privacy CA when attesting to a TCPA TPM identity.
- tpm_model is the means of referencing the type of implementation of protected capabilities and shielded locations. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.
- tpm_distributed_validation is a convenient immediate reference to the security properties of the implementation of protected capabilities and shielded locations. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.
- Access to the TPM_ENDORSEMENT_CREDENTIAL must be restricted to entities that have a need to know. This is for reasons of privacy.

*End of informative comment*

**Description**

```
struct TPM_ENDORSEMENT_CREDENTIAL = {
      BYTE              label = "TCPA Trusted Platform Module Endorsement"
      TCPA_PUBKEY       public_endorsement_key
      REFERENCE         tpm_model
      REFERENCE         tpm_distributed_validation
      REFERENCE         tpme_reference
      TCPA_VERSION      TCPA_VERSION
      SIGNATURE         signature_value}
```

This is an abstract definition, section 9.5.1 contains the concrete representation.

**Parameters**

| Type | Name | Description |
|---|---|---|

| | | |
|---|---|---|
| BYTE | label | This SHALL be the ASCII characters "TCPA Trusted Platform Module Endorsement" |
| TCPA_PUBKEY | public_endorsement_key | This SHALL be the PUBEK returned by a TPM_CreateEndorsementKeyPair command. |
| REFERENCE | tpm_model | This SHALL be a reference to the type of implementation of protected capabilities and shielded locations that created the PUBEK, plus a reference to the identity of the manufacturer of that implementation. |
| REFERENCE | tpm_distributed_validation | This SHALL be a reference to fields that indicate the security qualities of the implementation of protected capabilities and shielded locations that created the PUBEK. |
| REFERENCE | tpme_reference | This SHALL be an unambiguous indication of the identity of the (TPM) entity that attests that the implementation of protected capabilities and shielded locations conforms to the TCPA specification. |
| TCPA_VERSION | TCPA_VERSION | This SHALL be the version specified in section 4.5. |
| SIGNATURE | signature_value | This SHALL be the signature over all previous fields in TPM_ENDORSEMENT_CREDENTIAL, using the private key of the tpme-reference. |

When an entity presents evidence to a Privacy CA that an implementation of protected capabilities and shielded locations conforms to the TCPA specification, that evidence SHALL include the data in the data structure TPM_ENDORSEMENT_CREDENTIAL.

A (TPME) entity SHALL NOT create the data structure TPM_ENDORSEMENT_CREDENTIAL unless the entity is satisfied that the PUBEK referenced in TPM_ENDORSEMENT_CREDENTIAL was returned in response to a TPM_CreateEndorsementKeyPair command by an implementation of protected capabilities and shielded locations that meets the TCPA specification.

If the data structure TPM_ENDORSEMENT_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

### 4.32.2 Evidence of Platform Endorsement

*Start of informative comment:*

The purpose of platform credential is to provide evidence that a platform correctly incorporates an implementation of the protected capabilities and shielded locations of a TCPA Subsystem.

Platform_credential is an attestation that a platform contains a genuine TCPA Subsystem. Platform_credential contains information that a Privacy CA may use in judging whether the Privacy CA will attest to an identity of that TCPA Subsystem. Platform_credential contains information that the Privacy CA must use in attesting to an identity of that TCPA Trusted Platform Subsystem.

Platform_credential is tagged with TCPA_VERSION so as to indicate the version of the capability that created the PUBEK at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

- TPM-reference is the means of referencing the specific implementation of protected capabilities and shielded locations that is incorporated into the platform. It will be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity.
- The conformance-credential contains a set of conformance UIDs that unambiguously indicate the conformance to the TCPA specification of the TPM that is incorporated into the platform. These UIDs are the "tpm-protection-profile" and "tpm-security-target". The conformance credential also contains a set of conformance UIDs that unambiguously indicate the conformance to the TCPA specification of the means by which the platform incorporates an implementation of the TPM, the implementation of the root-of-trust-for-measurement, and the means by which the platform incorporates an implementation of the root-of-trust-for-measurement. These UIDs are the "foundation-protection-profile" and "foundation-security-target". All these UIDs will be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity.
- "TCPA Trusted Platform Endorsement" identifies a data structure as platform_credential and enables the Platform Entity (PE) to sign the data with a key that is not exclusively reserved for signing platform_credential.
- PE-reference is the means of referencing the PE. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity.
- platform_model is the means of referencing the type of platform. The reference includes the implementation of TCPA foundations in the platform. The foundations include the root-of-trust-for measurement that is incorporated into the platform, the method of incorporation of the RTM, and the method of incorporation of the TPM. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.
- platform_distributed_validation is a convenient immediate reference to the security properties of the platform. The reference includes the implementation of TCPA foundations in the platform. The foundations include the RTM that is incorporated into the platform, the method of incorporation of the RTM, and the method of incorporation of the TPM. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.

Access to the platform_credential must be restricted to entities that have a "need to know." This is for reasons of privacy.

*End of informative comment.*

**Description**

When an entity presents evidence to a Privacy CA that a platform conforms to the TCPA specification, that evidence SHALL include the data in the data structure platform_credential.

An entity (PE) SHALL NOT create the data structure platform_credential unless the entity is satisfied that the platform conforms to the conformance credential referenced inside platform_credential and contains the TPM referenced inside platform_credential.

**Definition**

```
struct PLATFORM_CREDENTIAL ={
      ASCII_STRING        "TCPA Trusted Platform Endorsement"
      REFERENCE           tpm-credential-reference
      REFERENCE           conformance-credential-reference
      REFERENCE           platform_TBB
      REFERENCE           platform_distributed_validation
      REFERENCE           pe-reference
      TCPA_VERSION        TCPA_VERSION
      SIGNATURE           signature_value}
```

This is an abstract definition, section 9.5.2 contains the concrete representation.

**Parameters**

| Type | Name | Description |
|---|---|---|
| ASCII_STRING | "TCPA Trusted Platform Endorsement" | This SHALL be the ASCII string "TCPA Trusted Platform Endorsement" |
| REFERENCE | tpm-credential-reference | This SHALL be an unambiguous indication of the endorsement credential of the TPM incorporated into the platform. |
| REFERENCE | conformance-credential-reference | This SHALL be an unambiguous indication of the conformance UIDs that attest that the design of the platform conforms to the TCPA specification. |
| REFERENCE | platform_TBB | This SHALL be a reference to the type of the platform, including the TCPA foundations in the platform, plus a reference to the identity of the manufacturer of that platform. |
| REFERENCE | platform_distributed_validation | This SHALL be fields that indicate the general security qualities of the platform. |
| REFERENCE | pe-reference | This SHALL be an unambiguous indication of the identity of the (platform) entity that attests to the design and construction of the platform. |
| TCPA_VERSION | TCPA_VERSION | This SHALL be the version specified in section 4.5. |
| SIGNATURE | signature_value | This SHALL be the signature over all previous fields in platform_credential, using the private key of the pe-reference. |

If the data structure platform_credential is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

### 4.32.3 Evidence of Platform Conformance

*Start of informative comment:*

The purpose of conformance credential is to provide evidence that the design of the Subsystem in a platform correctly conforms to the TCPA specification, and that the design of the method of incorporation of the Subsystem in the platform correctly conforms to the TCPA specification.

Conformance credential is an attestation that the overall design of a platform satisfies the TCPA specification. Conformance credential contains information that a Privacy CA may use in judging whether the Privacy CA will attest to an identity of that TCPA Subsystem. Conformance credential contains information that the Privacy CA must use in attesting to an identity of that TCPA Trusted Platform Subsystem.

Conformance credential is tagged with TCPA_VERSION so as to indicate the version of the capability that created the PUBEK at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

Conformance credential contains identifiers (UIDs) that indicate the protection profile and the security target of both the TPM and the RTM, and the methods by which they are incorporated into the platform.

*End of informative comment.*

**Description**

When an entity presents evidence to a Privacy CA that a platform conforms to the TCPA specification, that evidence SHALL include the data in the data structure conformance_credential.

A (conformance) entity SHALL NOT create the data structure conformance_credential unless the entity is satisfied that the design of both the Subsystem and its incorporation into the platform are accurately and unambiguously represented by the information in conformance_credential.

```
typedef struct CONFORMANCE_CREDENTIAL ={
      ASCII_STRING       "TCPA Conformance Credential"
      CONFORM_UID        tpm_pp
      CONFORM_UID        tpm_st
      CONFORM_UID        foundation_pp
      CONFORM_UID        foundation_st
      REFERENCE          ce_reference
      TCPA_VERSION       TCPA_VERSION
      SIGNATURE          signature
}
```

This is an abstract definition; section 9.5.3 contains the concrete representation.

**Parameters**

| Type | Name | Description |
|---|---|---|
| ASCII_STRING | "TCPA Conformance Credential" | This SHALL be the ASCII string "TCPA Conformance Credential" |
| CONFORM_UID | tpm_pp | This SHALL be the UID that unambiguously identifies the protection profile of the TPM |
| CONFORM_UID | tpm_st | This SHALL be the UID that unambiguously identifies the security target of the TPM |
| CONFORM_UID | foundation_pp | This SHALL be the UID that unambiguously identifies the protection profile of the TCPA foundations in the platform. |
| CONFORM_UID | foundation_st | This SHALL be the UID that unambiguously identifies the security target of the TCPA |

| | | |
|---|---|---|
| | | identifies the security target of the TCPA foundations in the platform. |
| REFERENCE | ce_reference | This SHALL be an unambiguous indication of the identity of the (Conformance) entity that attests to the overall design of the platform. |
| TCPA_VERSION | TCPA_VERSION | This SHALL be the version specified in section 4.5. |
| SIGNATURE | signature_value | This SHALL be the signature over all previous fields in CONFORMANCE_CREDENTIAL, using the private key of the ce_reference. |

### 4.32.4 TCPA Validation Data

*Start of informative comment:*

The purpose of TCPA Validation Data is to state the values of integrity metrics that should be obtained when the component described by the validation data is working properly.

TCPA Validation Data identifies a data structure as validation data and enables the PE to sign the data with a key that is not exclusively reserved for signing validation data.

*End of informative comment.*

All components that influence the software environment in a platform SHOULD have corresponding validation data.

The representation of a component SHALL reflect the way that the component influences the software environment in a platform. All representations SHALL include a description of the manufacturer, the common name of the component, the version of the component, and a field that describes the security qualities of the component.

The representation of a component SHALL NOT in any way provide information that exposes the identity of a specific component.

The validation data of a component SHALL be validation_data

**IDL Description**

```
typedef struct VALIDATION_DATA ={
        ASCII_STRING            "TCPA Validation Data"
        ASCII_STRING            component_manufacturer,
        ASCII_STRING            component_name,
        ASCII_STRING            component_version,
        DIGEST                  instruction_digest,
        REFERENCE               component_distributed_validation,
        REFERENCE               ve_reference,
        TCPA_VERSION            TCPA_VERSION,
        SIGNATURE               validation_data_signature_value}
```

This is an abstract definition; section 9.5.4 contains the concrete representation.

**Parameters**

| Type | Name | Description |
|---|---|---|
| ASCII_STRING | "TCPA Validation Data" | This SHALL be the ASCII string "TCPA Validation Data." |
| ASCII_STRING | component_manufacturer | This SHALL be an ASCII string stating the name of the manufacturer of the component. |
| ASCII_STRING | component_name | This SHALL be an ASCII string stating the common name of the component. |
| ASCII_STRING | component_version | This SHALL be an ASCII string stating the version of the component. |
| DIGEST | instruction_digest | This SHALL be a digest of any instructions in the component that are intended to execute on the main computing engine of the platform. |
| REFERENCE | component_distributed_validation | This SHALL be a convenient immediate reference to the security properties of the |

| | | |
|---|---|---|
| | | reference to the security properties of the component. |
| REFERENCE | ve_reference | This SHALL be an unambiguous indication of the identity of the (validation) entity that attests to the validation data. |
| TCPA_VERSION | TCPA_VERSION | This SHALL be the version specified in section 4.5. |
| SIGNATURE | validation_data_signature_value | This SHALL be the result of signing all fields (except this field) in VALIDATION_DATA using the signature (private) key of VE_reference. |

### 4.32.5 Evidence of Trusted Platform Module Identity

*Start of informative comment*

The data in TPM_IDENTITY_CREDENTIAL is presented whenever an entity requires proof that an anonymous identity belongs to a genuine TCPA Subsystem.

TPM_IDENTITY_CREDENTIAL may be accompanied by other data, depending upon circumstances. When presented in response to an integrity challenge, it may be accompanied by conventional certificates and validation data, for example.

TPM_IDENTITY_CREDENTIAL is tagged with TCPA_VERSION so as to indicate the version of the capability that created the identity key at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

The phrase "TCPA Trusted Platform Module identity" identifies a data structure as a Trusted Platform Module identity and enables the Privacy CA to sign the data with a key that is not exclusively reserved for signing TPM identities.

Access to the TPM_IDENTITY_CREDENTIAL must be restricted to entities that have a "need to know". This is for reasons of privacy.

*End of informative comment*

**Description**

When an entity presents evidence that an identity belongs to a Subsystem, that evidence SHALL include the data in the data structure TPM_IDENTITY_CREDENTIAL.

```
struct TPM_IDENTITY_CREDENTIAL ={
      ASCII_STRING      "TCPA Trusted Platform Identity"
      UNICODE           identityLabel
      TCPA_PUBKEY       identityPubKey
      REFERENCE         tpm_model
      REFERENCE         tpm_distributed_validation
      CONFORM_UID       tpm_pp
      CONFORM_UID       tpm_st
      REFERENCE         platform_model
      REFERENCE         platform_distributed_validation
      CONFORM_UID       foundation_pp
      CONFORM_UID       foundation_st
      REFERENCE         p-ca_reference
      TCPA_VERSION      TCPA_VERSION
      SIGNATURE         signature_value}
```

This is an abstract definition; section 9.5.5 contains the concrete representation.

**Parameters**

| Type | Name | Description |
|---|---|---|
| ASCII_STRING | "TCPA Trusted Platform Module Identity" | This SHALL be the ASCII string "TCPA Trusted Platform Identity." |
| UNICODE | identityLabel | This SHALL be a textual string associated with the TPM identity. |
| TCPA_PUBKEY | identityPubKey | This SHALL be a public key associated with the TPM identity. |
| REFERENCE | tpm_model | This SHALL be a reference to the type of TPM in the platform, plus a reference to the identity of the manufacturer of TPM. |
| REFERENCE | tpm_distributed_validation | This SHALL be fields that indicate the security qualities of the TPM in the platform. |
| CONFORM_UID | tpm_pp | This SHALL be the UID that unambiguously identifies the protection profile of the TPM |
| CONFORM_UID | tpm_st | This SHALL be the UID that unambiguously identifies the security target of the TPM |
| REFERENCE | platform_model | This SHALL be a reference to the type of the platform, including the TCPA foundations in the platform, plus a reference to the identity of the manufacturer of that platform. |
| REFERENCE | platform_distributed_validation | This SHALL be fields that indicate the security qualities of the platform. |
| CONFORM_UID | foundation_pp | This SHALL be the UID that unambiguously identifies the protection profile of the TCPA foundations in the platform. |
| CONFORM_UID | foundation_st | This SHALL be the UID that unambiguously identifies the security target of the TCPA foundations in the platform. |
| REFERENCE | p-ca_reference | This SHALL be an unambiguous indication of the identity of the (Privacy CA) entity that attests to the TPM identity. |
| TCPA_VERSION | TCPA_VERSION | This SHALL be the version specified in section 4.5. |
| SIGNATURE | signature_value | This SHALL be the signature over all previous fields in TPM_IDENTITY_CREDENTIAL, using the private key of the p-ca_reference. |

If the data structure TPM_IDENTITY_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

## 4.33 Command Ordinals

*Start of informative comment:*

The command ordinals provide the index value for each command. The following list contains both the index value and a flag that indicates the default audit state of the command. The commands selected to be audited by default are those that substantially change the state of the TPM and/or the protected storage hierarchy.

TCPA commands are divided into three classes: Protected/Unprotected, Non-Connection/Connection related, and TCPA/Vendor.

*End of informative comment.*

Ordinals are 32 bit values. The upper byte contains values that serve as flag indicators, the next byte contains values indicating what committee designated the ordinal, and the final two bytes contain the Command Ordinal Index.

```
   3                   2                   1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|P|C|V| Reserved|     Purview     |    Command Ordinal Index      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where:

- P is Protected/Unprotected command. When 0 the command is a Protected command, when 1 the command is an Unprotected command.
- C is Non-Connection/Connection related command. When 0 this command passes through to either the protected (TPM) or unprotected (TSS) components.
- V is TCPA/Vendor command. When 0 the command is TCPA defined, when 1 the command is vendor defined.
- All reserved area bits are set to 0.

The following masks are created to allow for the quick definition of the commands

| Value | Event Name | Comments |
|---|---|---|
| 0x00000000 | TCPA_PROTECTED_COMMAND | TPM protected command, specified in main specification |
| 0x80000000 | TCPA_UNPROTECTED_COMMAND | TSS command, specified in the TSS specification |
| 0x40000000 | TCPA_CONNECTION_COMMAND | TSC command, protected connection commands are specified in the main specification. Unprotected connection commands are specified in the TSS. |
| 0x20000000 | TCPA_VENDOR_COMMAND | Command that is vendor specific for a given TPM or TSS. |

The following Purviews have been defined:

| Value | Event Name | Comments |
|---|---|---|
| 0x00 | TCPA_MAIN | Command is from the main specification |
| 0x01 | TCPA_PC | Command is specific to the PC |
| 0x02 | TCPA_PDA | Command is specific to a PDA |
| 0x03 | TCPA_CELL_PHONE | Command is specific to a cell phone |

Combinations for the main specification would be

| Value | Event Name |
|---|---|
| TCPA_PROTECTED_COMMAND \| TCPA_MAIN | TCPA_PROTECTED_ORDINAL |
| TCPA_UNPROTECTED_COMMAND \| TCPA_MAIN | TCPA_UNPROTECTED_ORDINAL |
| TCPA_CONNECTION_COMMAND \| TCPA_MAIN | TCPA_CONNECTION_ORDINAL |

If a command is tagged from the audit column the default state is that use of that command SHALL be audited. Otherwise, the default state is that use of that command SHALL NOT be audited.

| | TCPA_PROTECTED_ORDINAL + | Audit |
|---|---|---|
| TPM_ORD_OIAP | 10 | |
| TPM_ORD_OSAP | 11 | |
| TPM_ORD_ChangeAuth | 12 | |
| TPM_ORD_TakeOwnership | 13 | x |
| TPM_ORD_ChangeAuthAsymStart | 14 | |
| TPM_ORD_ChangeAuthAsymFinish | 15 | |
| TPM_ORD_ChangeAuthOwner | 16 | x |
| | | |
| TPM_ORD_Extend | 20 | |
| TPM_ORD_PcrRead | 21 | |
| TPM_ORD_Quote | 22 | |
| TPM_ORD_Seal | 23 | x |
| TPM_ORD_Unseal | 24 | |
| TPM_ORD_DirWriteAuth | 25 | x |
| TPM_ORD_DirRead | 26 | |
| | | |
| TPM_ORD_UnBind | 30 | |
| TPM_ORD_CreateWrapKey | 31 | x |
| TPM_ORD_LoadKey | 32 | |
| TPM_ORD_GetPubKey | 33 | |
| TPM_ORD_EvictKey | 34 | |
| | | |
| TPM_ORD_CreateMigrationBlob | 40 | x |
| TPM_ORD_ReWrapKey | 41 | |
| TPM_ORD_ConvertMigrationBlob | 42 | x |
| TPM_ORD_AuthorizeMigrationKey | 43 | x |
| TPM_ORD_CreateMaintenanceArchive | 44 | x |
| TPM_ORD_LoadMaintenanceArchive | 45 | x |
| TPM_ORD_KillMaintenanceFeature | 46 | x |
| TPM_ORD_LoadManuMaintPub | 47 | x |

| | | |
|---|---|---|
| TPM_ORD_ReadManuMaintPub | 48 | x |
| | | |
| TPM_ORD_CertifyKey | 50 | |
| | | |
| TPM_ORD_Sign | 60 | |
| | | |
| TPM_ORD_GetRandom | 70 | |
| TPM_ORD_StirRandom | 71 | |
| | | |
| TPM_ORD_SelfTestFull | 80 | |
| TPM_ORD_SelfTestStartup | 81 | |
| TPM_ORD_CertifySelfTest | 82 | |
| TPM_ORD_ContinueSelfTest | 83 | |
| TPM_ORD_GetTestResult | 84 | |
| | | |
| TPM_ORD_Reset | 90 | x |
| TPM_ORD_OwnerClear | 91 | x |
| TPM_ORD_DisableOwnerClear | 92 | x |
| TPM_ORD_ForceClear | 93 | x |
| TPM_ORD_DisableForceClear | 94 | x |
| | | |
| TPM_ORD_GetCapabilitySigned | 100 | |
| TPM_ORD_GetCapability | 101 | |
| TPM_ORD_GetCapabilityOwner | 102 | |
| | | |
| TPM_ORD_OwnerSetDisable | 110 | x |
| TPM_ORD_PhysicalEnable | 111 | x |
| TPM_ORD_PhysicalDisable | 112 | x |
| TPM_ORD_SetOwnerInstall | 113 | x |
| TPM_ORD_PhysicalSetDeactivated | 114 | x |
| TPM_ORD_SetTempDeactivated | 115 | x |
| | | |
| TPM_ORD_CreateEndorsementKeyPair | 120 | x |
| TPM_ORD_MakeIdentity | 121 | x |
| TPM_ORD_ActivateIdentity | 122 | x |
| TPM_ORD_ReadPubek | 124 | x |
| TPM_ORD_OwnerReadPubek | 125 | x |
| TPM_ORD_DisablePubekRead | 126 | x |
| | | |
| TPM_ORD_GetAuditEvent | 130 | x |
| TPM_ORD_GetAuditEventSigned | 131 | x |
| | | |
| TPM_ORD_GetOrdinalAuditStatus | 140 | |
| TPM_ORD_SetOrdinalAuditStatus | 141 | x |
| | | |
| TPM_ORD_Terminate_Handle | 150 | |
| TPM_ORD_Init | 151 | x |
| TPM_ORD_SaveState | 152 | x |
| TPM_ORD_Startup | 153 | x |
| TPM_ORD_SetRedirection | 154 | x |
| | | |
| TPM_ORD_SHA1Start | 160 | |
| TPM_ORD_SHA1Update | 161 | |
| TPM_ORD_SHA1Complete | 162 | |

| | | |
|---|---|---|
| TPM_ORD_SHA1CompleteExtend | 163 | |
| | | |
| TPM_ORD_FieldUpgrade | 170 | |
| | | |
| TPM_ORD_SaveKeyContext | 180 | |
| TPM_ORD_LoadKeyContext | 181 | |
| TPM_ORD_SaveAuthContext | 182 | |
| TPM_ORD_LoadAuthContext | 183 | |

The connection commands manage the TPM's connection to the TBB.

| | TCPA_CONNECTION_ORDINAL + |
|---|---|
| TSC_ORD_PhysicalPresence | 10 |

# 5. Authorization and Ownership

## 5.1 Introduction

*Start of informative comment:*

The purpose of the authorization mechanism is to authenticate an owner and to authorize use of an entity. The basic premise is to prove knowledge of a shared secret. This shared secret is the authorization data.

Authorization data is available for the TPM Owner and each entity (keys, for example) that the TPM controls. The authorization data for the TPM Owner and the SRK are held within the TPM itself and the authorization data for other entities are held with the entity.

The TPM Owner authorization data allows the Owner to prove ownership of the TPM. Proving ownership of the TPM does not immediately allow all operations – the TPM Owner is not a "super user" and additional authorization data must be provided for each entity or operation that has protection.

The TPM treats knowledge of the authorization data as complete proof of ownership of the entity. No other checks are necessary. The requestor (any entity that wishes to execute a command on the TPM or use a specific entity) may have additional protections and requirements where he or she (or it) saves the authorization data; however, the TPM places no additional requirements.

There are two protocols to securely pass a proof of knowledge of authorization data from requestor to TPM: the "Object-Independent Authorization Protocol" (OI-AP) and the "Object-Specific Authorization Protocol" (OS-AP). The OI-AP supports multiple authorization sessions for arbitrary entities. The OS-AP supports an authentication session for a single entity and enables the confidential transmission of new authorization information. That new authorization information is inserted by the "Authorization Data Insertion Protocol" (ADIP) during the creation of an entity. The "Authorization Data Change Protocol" (ADCP) and the "Asymmetric Authorization Change Protocol" (AACP) allow the changing of the authorization data for an entity. The protocol definitions allow expansion of protocol types to additional TCPA required protocols and vendor specific protocols.

The protocols use a "rolling nonce" paradigm. This requires that a nonce from one side be in use only for a message and its reply. For instance, the TPM would create a nonce and send that on a reply. The requestor would receive that nonce and then include it in the next request. The TPM would validate that the correct nonce was in the request and then create a new nonce for the reply. This mechanism is in place to prevent replay attacks and man-in-the-middle attacks.

The basic protocols do not provide long-term protection of authorization data that is the hash of a password or other low-entropy entities. The TPM designer and application writer must supply additional protocols if protection of these types of data is necessary.

The design criterion of the protocols is to allow for ownership authentication, command and parameter authentication and prevent replay and man-in-the-middle attacks.

The passing of the authorization data, nonces and other parameters must follow specific guidelines so that commands coming from different computer architectures will interoperate properly.

*End of informative comment.*

All entity authorizations requiring authorization MUST use the authorization data protocols.

The TPM MUST support the OI-AP and the OS-AP which enable proof of knowledge of authorization data while maintaining the secrecy of that authorization data.

The TPM MUST support the ADIP that inserts the authorization during entity creation.

The TPM MUST support the ADCP and AACP which allow for the changing of authorization data.

The TPM MUST support TPM_Terminate_Handle which forces the termination of a session.

The TPM MAY support additional protocols to authenticate, insert and change authorization data.

The TPM MUST support the ability to calculate a HMAC in order to verify authorization data independent of the source or transmission mechanism. The TPM MUST calculate the HMAC digest according to section 8.6. The TPM MUST NOT perform the HMAC calculation for a returning message when the authorization for the command fails or the command fails for any other reason.

If a command has more than one authorization value, each authorization session MUST use the same SHA-1 parameter digest (<paramDigest> from Sect. 4.4.2) plus its respective authorization setup parameters (nonces, authHandles, etc) in the HMAC calculation. For example, the capability 9.3.1TPM_MakeIdentity requires authorization from both the TPM Owner and from the SRK owner. So the authentication information "TpmOwnerAuth" and "SrkAuth" are each calculated over all parameters tagged with an 'S' subscript in the definition of TPM_MakeIdentity.

All commands that use keys normally include at least one authorization session in the input parameters. If AuthDataUsage is set to TPM_AUTH_NEVER for that key, then the command does not need to be authorized. To implement this, the 5 authorization parameters at the end of the input parameter list should be removed and the tag value (first parameter) changed from TPM_TAG_RQU_AUTH1_COMMAND to TPM_TAG_RQU_COMMAND.

When an incoming command includes an authorization session but the authorized key has AuthDataUsage set to NEVER the TPM MUST perform the following:

- If the value of the command tag is TPM_TAG_RQU_AUTH1_COMMAND the TPM will compute the authorization based on the value store in the authorization location within the key, IGNORING the state of the AuthDataUsage flag.
- Users may choose to use a well-known value for the authorization data when setting AuthDataUsage to NEVER.

For commands that normally have 2 authorization sessions, if the tag specifies only one in the parameter array, then the first session listed is ignored (authDataUsage must be NEVER for this key) and the incoming session data is used for the second auth session in the list.

### 5.1.1 Tag Usage

This table summarizes what can be the tag with a given TPM command.

| Section | Name | Tag: AUTH2_COMMAND | Tag: AUTH1_COMMAND | Tag: RQU_COMMAND |
|---|---|---|---|---|
| 5.6.1 | TPM_ChangeAuth | x | | |
| 5.6.2 | TPM_ChangeAuthOwner | | x | |
| 5.7.1 | TPM_ChangeAuthAsymStart | | x | x |
| 5.7.2 | TPM_ChangeAuthAsymFinish | | x | x |
| 5.11.1 | TPM_TakeOwnership | | x | |
| 6.3.3 | TPM_Quote | | x | x |
| 6.3.4 | TPM_DirWriteAuth | | x | |
| 7.2.1 | TPM_Seal | | x | |
| 7.2.2 | TPM_Unseal | x | x | |
| 7.2.4 | TPM_UnBind | | x | x |
| 7.2.5 | TPM_CreateWrapKey | | x | |
| 7.2.8 | TPM_LoadKey | | x | x |
| 7.2.10 | TPM_GetPubKey | | x | x |
| 7.2.11 | TPM_CreateMigrationBlob | x | x | x |
| 0 | TPM_ConvertMigrationBlob | | x | x |
| 7.2.13 | TPM_AuthorizeMigrationKey | | x | |
| 7.3.1 | TPM_CreateMaintenanceArchive | | x | |
| 7.3.2 | TPM_LoadMaintenanceArchive | | x | |
| 7.3.3 | TPM_KillMaintenanceFeature | | x | |
| 8.3.1 | TPM_CertifyKey | x | x | x |
| 8.7.1 | TPM_Sign | | x | x |
| 8.9.2 | TPM_CertifySelfTest | | x | x |
| 0 | TPM_OwnerClear | | x | |
| 8.10.6 | TPM_DisableOwnerClear | | x | |
| 8.11.2 | TPM_GetCapabilitySigned | | x | x |
| 8.11.3 | TPM_GetCapabilityOwner | | x | |
| 8.12.2 | TPM_GetAuditEventSigned | | x | x |
| 8.12.3 | TPM_SetOrdinalAuditStatus | | x | |
| 8.14.1 | TPM_OwnerSetDisable | | x | |
| 8.17 | TPM_SetRedirection | | x | x |
| 9.2.3 | TPM_DisablePubekRead | | x | |
| 9.2.4 | TPM_OwnerReadPubek | | x | |
| 9.3.1 | TPM_MakeIdentity | x | x | |
| 9.3.4 | TPM_ActivateIdentity | x | x | |

## 5.2 Authorization protocols

***Start of informative comment:***

The TPM provides two protocols for authorizing the use of entities without revealing the authorization data on the network or the connection to the TPM. In both cases, the protocol exchanges nonce-data so that both sides of the transaction can compute a hash using shared secrets and nonce-data. Each side generates the hash value and can compare to the value transmitted. Network listeners cannot directly infer the authorization data from the hashed objects sent over the network.

The first protocol is the "Object-Independent Authorization Protocol" (OI-AP), which allows the exchange of nonces with a specific TPM. Once an OI-AP session is established, its nonces can be used to authorize the use any entity managed by the TPM. The session can live indefinitely until either party request the session termination. The TPM_OIAP function starts the OI-AP session.

The second protocol is the "Object Specific Authorization Protocol" (OS-AP). The OS-AP allows establishment of an authentication session for a single entity. The session creates nonces that can authorize multiple commands without additional session-establishment overhead, but is bound to a specific entity. The TPM_OSAP command starts the OS-AP session. The TPM_OSAP specifies the entity to which the authorization is bound.

Most commands allow either form of authorization protocol. In general, however, the OI-AP is preferred – it is more generally useful because it allows usage of the same session to provide authorization for different entities. The OS-AP is, however, necessary for operations that set or reset authorization data.

OI-AP sessions were designed for reasons of efficiency; only one setup process is required for potentially many authorizations.

An OS-AP session is doubly efficient because only one setup process is required for potentially many authorization calculations and the entity authorization secret is required only once. This minimizes exposure of the authorization secret and can minimize human interaction in the case where a person supplies the authorization information. The disadvantage of the OS-AP is that a distinct session needs to be setup for each entity that requires authorization. The OS-AP creates an ephemeral secret that is used throughout the session instead of the entity authorization secret. The ephemeral secret can be used to provide confidentiality for the introduction of new authorization data during the creation of new entities. Termination of the OS-AP occurs in two ways. Either side can request session termination (as usual) but the TPM forces the termination of an OS-AP session after use of the ephemeral secret for the introduction of new authorization data.

For both the OS-AP and the OI-AP, session setup is independent of the commands that are authorized. In the case of OI-AP, the requestor sends the TPM_OIAP command, and with the response generated by the TPM, can immediately begin authorizing object actions. The OS-AP is very similar, and starts with the requestor sending a TPM_OSAP operation, naming the entity to which the authorization session should be bound.

Both session types use a "rolling nonce" paradigm. This means that the TPM creates a new nonce value each time the TPM uses the session for a HMAC calculation.

Note that some operations involve the use of two authorization elements (for example, UNSEAL requires the authorization data of the object itself and authorization data of the object's parent). In this case, two separate sessions are required. It is not possible to use one session for both purposes.

For the purposes of the informative comments for the individual protocols, the following example command will be used, named TPM_Example. Not that this command has a single authorization session, and that the authorization secret is the auth value stored with some key. Commands in this document have from 0 to 2 authorization sessions.

Some commands within this document use secrets other than the auth value in a key. Two examples would be owner authorized commands, or commands using keyMigration as the secret. In this case key.usageAuth in the examples below would be replaced with ownerAuth, KeyMigration or other secrets as necessary. In all cases, the secret used to compute the authorization digest is noted in the description for the actual digest parameter within the command parameter lists.

**Incoming Operands and Sizes**

| *Param* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *Sz* | *#* | *Sz* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_Example |
| 4 | 4 | | | TCPA_KEY_HANDLE | keyHandle | Handle of a loaded key |
| 5 | 1 | 2s | 1 | BOOL | inArgOne | The first input argument |
| 6 | 20 | 3s | 20 | UINT32 | inArgTwo | The second input argument |
| 7 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TCPA_AUTHDATA | inAuth | The authorization digest for inputs and keyHandle. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| *Param* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *Sz* | *#* | *Sz* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3 |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_Example |
| 4 | 4 | 3s | 4 | UINT32 | outArgOne | Output argument |
| 5 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth. |

*End of informative comment.*

### 5.2.1 OI-AP description

***Start of informative comment:***

The purpose of this section is to illustrate the OI-AP without regard to a specific command. OI-AP uses the TPM_OIAP command to create the authorization session. See Section 5.2.2 for the TPM_OIAP description.

Assume that a TPM user wishes to send command TPM_Example. This is an authorized command that uses the key denoted by keyHandle. The user must know the authorization data for keyHandle (key.usageAuth) as this is the entity that requires authorization and this secret is used in the authorization calculation. Let us assume for this example that the caller of TPM_Example does not need to authorize the use of keyHandle for more than one command. This use model points to the selection of the OI-AP as the authorization protocol.

For the TPM_Example command, the inAuth parameter provides the authorization to execute the command. The following table shows the commands executed, the parameters created and the wire formats of all of the information.

<inParamDigest> is the result of the following calculation: SHA1(ordinal, inArgOne, inArgTwo). <outParamDigest> is the result of the following calculation: SHA1(returnCode, ordinal, outArgOne). inAuthSetupParams refers to the following parameters, in this order: authHandle, authLastNonceEven, nonceOdd, continueAuthSession. OutAuthSetupParams refers to the following parameters, in this order: authHandle, nonceEven, nonceOdd, continueAuthSession.

There are two even nonces used to execute TPM_Example, the one generated as part of the TPM_OIAP command (labeled authLastNonceEven below) and the one generated with the output arguments of TPM_Example (labeled as nonceEven below).

| Caller | On the wire | Dir | TPM |
|---|---|---|---|
| Send TPM_OIAP | TPM_OIAP | → | • Create session<br>• Create authHandle<br>• Associate session and authHandle<br>• Generate authLastNonceEven<br>• Save authLastNonceEven with authHandle |
| Save authHandle, authLastNonceEven | authHandle, authLastNonceEven | ← | Returns |
| • Generate nonceOdd<br>• Compute inAuth = HMAC (key.usageAuth, inParamDigest, inAuthSetupParams)<br>• Save nonceOdd with authHandle | | | |
| Send TPM_Example | tag<br>paramSize<br>ordinal<br>inArgOne<br>inArgTwo<br>authHandle<br>nonceOdd<br>continueAuthSessio<br>inAuth | → | • TPM retrieves key.usageAuth (key must hav been previously loaded)<br>• Verify authHandle points to a valid session, mismatch returns TPM_E_INVALIDAUTH<br>• Retrieve authLastNonceEven from internal session storage<br>• HM = HMAC (key.usageAuth, inParamDige inAuthSetupParams)<br>• Compare HM to inAuth. If they do not compare return with TPM_E_INVALIDAUTH<br>• Execute TPM_Example and create returnCode<br>• Generate nonceEven to replace authLastNonceEven in session<br>• Set resAuth = HMAC (key.usageAuth, outParamDigest, outAuthSetupParams) |
| • Save nonceEven<br>• HM = HMAC( key.usageAuth, outParamDigest, outAuthSetupParams)<br>• Compare HM to resAuth. This verifies returnCode and output parameters | tag<br>paramSize<br>returnCode<br>outArgOne<br>nonceEven<br>continueAuthSessio<br>resAuth | ← | • Return output parameters<br>• If continueAuthSession is FALSE then destro session |

Suppose now that the TPM user wishes to send another command using the same session. For the purposes of this example, we will assume that the same ordinal is to be used (TPM_Example) but that a different key (newKey) with its own secret (newKey.usageAuth) is to be operated on. To re-use the previous session, the continueAuthSession output boolean must be TRUE.

The following table shows the command execution, the parameters created and the wire formats of all of the information.

In this case, authLastNonceEven is the nonceEven value returned by the TPM with the output parameters from the first execution of TPM_Example.

| Caller | On the wire | Dir | TPM |
|---|---|---|---|
| • Generate nonceOdd<br>• Compute inAuth = HMAC (newKey.usageAuth, inParamDigest, inAuthSetupParams)<br>• Save nonceOdd with authHandle | | | |
| • Send TPM_Example | tag<br>paramSize<br>ordinal<br>inArgOne<br>inArgTwo<br>nonceOdd<br>continueAuthSession<br>inAuth | → | • TPM retrieves newKey.usageAuth (newKey must have been previously loaded)<br>• Retrieve authLastNonceEven from internal session storage<br>• HM = HMAC (newKey.usageAuth, inParamDigest, inAuthSetupParams)<br>• Compare HM to inAuth. If they do not compare return with TPM_E_INVALIDAUTH<br>• Execute TPM_Example and create returnCode<br>• Generate nonceEven to replace authLastNonceEven in session<br>• Set resAuth = HMAC(newKey.usageAut outParamDigest, outAuthSetupParams) |
| • Save nonceEven<br>• HM = HMAC( newKey.usageAuth, outParamDigest, outAuthSetupParams)<br>• Compare HM to resAut This verifies returnCod and output parameters | tag<br>paramSize<br>returnCode<br>outArgOne<br>nonceEven<br>continueAuthSession<br>resAuth | ← | • Return output parameters<br>• If continueAuthSession is FALSE then destroy session |

The TPM user could then use the session for further authorization sessions. Suppose, however, that the TPM user no longer requires the authorization session. There are three possibilities in this case:

- The user issues a TPM_Terminate_Handle command to the TPM (section 5.3).
- The input argument continueAuthSession can be set to FALSE for the last command. In this case, the output continueAuthSession value will be FALSE.
- In some cases, the TPM automatically terminates the authorization session regardless of the input value of continueAuthSession. In this case as well, the output continueAuthSession value will be FALSE.

When an authorization session is terminated for any reason, the TPM invalidates the session's handle and terminates the session's thread (releases all resources allocated to the session).

*End of informative comment*

### 5.2.2 TPM_OIAP

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_OIAP. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | TCPA_AUTHHANDLE | authHandle | Handle that TPM creates that points to the authorization state. |
| 5 | 20 | | | TCPA_NONCE | nonceEven | Nonce generated by TPM and associated with session. |

**Actions**

1. The TPM_OIAP command allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle and nonce.
2. The TPM has an internal limit as to the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. Internally the TPM will do the following:
   a) TPM allocates space to save handle, protocol identification, both nonces and any other information the TPM needs to manage the session.
   b) TPM generates authHandle and nonceEven, returns these to caller
4. On each subsequent use of the OIAP session the TPM MUST generate a new nonceEven value.

### 5.2.3 Authorization using an OI-AP session

*Start of informative comment:*

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OI-AP protocol.

Many commands use OI-AP authorization. The following description is therefore necessarily abstract.

*End of informative comment.*

**Actions**

perform the following actions:

1. The TPM MUST verify that the authorization handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TCPA_AUTHFAIL.
2. The TPM SHALL retrieve the latest version of the caller's nonce (nonceOdd) and continueAuthSession flag from the input parameter list, and store it in internal TPM memory with the authSession 'H'.
3. The TPM SHALL retrieve the latest version of the TPM's nonce stored with the authorization session H (authLastNonceEven) computed during the previously executed command.
4. The TPM MUST retrieve the secret authorization data (SecretE, say) of the target entity. The entity and its secret must have been previously loaded into the TPM.
5. The TPM SHALL perform a HMAC calculation using the entity secret data, ordinal, input command parameters and authorization parameters per section 4.4.2.
6. The TPM SHALL compare HM to the authorization value received in the input parameters. If they are different, the TPM returns the error code TCPA_AUTHFAIL. Otherwise, the TPM executes the command which (for this example) produces an output that requires authentication.
7. The TPM SHALL generate a nonce (nonceEven).
8. The TPM creates an HMAC digest to authenticate the return code, return values and authorization parameters to the same entity secret per section 4.4.2
9. The TPM returns the return code, output parameters, authorization parameters and authorization digest.
10. If the output continueUse flag is FALSE, then the TPM SHALL terminate the session. Future references to H will return an error.

### 5.2.4 OS-AP Description

*Start of informative comment:*

The OS-AP command creates an ephemeral secret to authenticate a session. The purpose of this section is to illustrate the OS-AP without regard to a specific command. See Section 5.2.5 for the TPM_OSAP description which is used to create this authorization session.

Assume that a TPM user wishes to send command TPM_Example. This is an authorized command that uses the key denoted by keyHandle. The user must know the authorization data for keyHandle (key.usageAuth) as this is the entity that requires authorization and this secret is used in the authorization calculation.

Let us assume for this example that the caller of TPM_Example needs to use this key multiple times but does not wish to obtain the key secret more than once. This might be the case if, for example, the usage authorization data were derived from a typed password. This use model points to the selection of the OS-AP as the authorization protocol.

For the TPM_Example command, the inAuth parameter provides the authorization to execute the command. The following table shows the commands executed, the parameters created and the wire formats of all of the information.

<inParamDigest> is the result of the following calculation: SHA1(ordinal, inArgOne, inArgTwo). <outParamDigest> is the result of the following calculation: SHA1(returnCode, ordinal, outArgOne). inAuthSetupParams refers to the following parameters, in this order: authLastNonceEven, nonceOdd, continueAuthSession. OutAuthSetupParams refers to the following parameters, in this order: nonceEven, nonceOdd, continueAuthSession

In addition to the two even nonces generated by the TPM (authLastNonceEven and nonceEven) that are used for TPM_OIAP, there is a third, labeled nonceEvenOSAP that is used to generate the shared secret. For every even nonce, there is also an odd nonce generated by the system.

| Caller | On the wire | Dir | TPM |
|---|---|---|---|
| Send TPM_OSAP | TPM_OSAP<br>keyHandle<br>nonceOddOSAP | → | • Create session & authHandle<br>• Generate authLastNonceEven<br>• Save authLastnonceEven with authHandle<br>• Generate nonceEvenOSAP<br>• Generate sharedSecret = HMAC(key.usageAuth, nonceEvenOSAP, nonceOddOSAP)<br>• Save keyHandle, sharedSecret with authHandle |
| • Save authHandle, authLastNonceEven<br>• Generate sharedSecret = HMAC(key.usageAuth, nonceEvenOSAP, nonceOddOSAP)<br>• Save sharedSecret | authHandle<br>authLastNonceEven<br>nonceEvenOSAP | ← | Returns |
| • Generate nonceOdd save with authHandle<br>• Compute inAuth = HMAC (sharedSecret, inParamDigest, inAuthSetupParams) | | | |
| Send TPM_Example | tag<br>paramSize<br>ordinal<br>inArgOne<br>inArgTwo<br>authHandle<br>nonceOdd<br>continueAuthSession<br>inAuth | → | • Verify authHandle points to a valid sessio mismatch returns TPM_AUTHFAIL<br>• Retrieve authLastNonceEven from intern session storage<br>• HM = HMAC (sharedSecret, inParamDigest, inAuthSetupParams)<br>• Compare HM to inAuth. If they do not compare return with TPM_AUTHFAIL<br>• Execute TPM_Example and create returnCode<br>• Generate nonceEven to replace authLastNonceEven in session<br>• Set resAuth = HMAC(sharedSecret, outParamDigest, outAuthSetupParams) |
| • Save nonceEven<br>• HM = HMAC( sharedSecret, outParamDigest, outAuthSetupParams)<br>• Compare HM to resAuth. This verifies returnCode and output parameters. | tag<br>paramSize<br>returnCode<br>outArgOne<br>nonceEven<br>continueAuthSession<br>resAuth | ← | • Return output parameters<br>• If continueAuthSession is FALSE then destroy session |

Suppose now that the TPM user wishes to send another command using the same session to operate on the same key. For the purposes of this example, we will assume that the same ordinal is to be used (TPM_Example). To re-use the previous session, the continueAuthSession output boolean must be TRUE.

The following table shows the command execution, the parameters created and the wire formats of all of the information.

In this case, authLastNonceEven is the nonceEven value returned by the TPM with the output parameters from the first execution of TPM_Example.

| Caller | On the wire | Dir | TPM |
|---|---|---|---|
| • Generate nonceOdd<br>• Compute inAuth = HMAC (sharedSecret, inParamDigest, inAuthSetupParams)<br>• Save nonceOdd with authHandle | | | |
| • Send TPM_Example | tag<br>paramSize<br>ordinal<br>inArgOne<br>inArgTwo<br>nonceOdd<br>continueAuthSession<br>inAuth | → | • Retrieve authLastNonceEven from internal session storage<br>• HM = HMAC (sharedSecret, inParamDigest, inAuthSetupParams)<br>• Compare HM to inAuth. If they do not compare return with TPM_AUTHFAIL<br>• Execute TPM_Example and create returnCode<br>• Generate nonceEven to replace authLastNonceEven in session<br>• Set resAuth = HMAC (sharedSecret, outParamDigest, outAuthSetupParams) |
| • Save nonceEven<br>• HM = HMAC( sharedSecret, outParamDigest, outAuthSetupParams)<br>• Compare HM to resAuth. This verifies returnCode and output parameters. | tag<br>paramSize<br>returnCode<br>outArgOne<br>nonceEven<br>continueAuthSession<br>resAuth | ← | • Return output parameters<br>• If continueAuthSession is FALSE then destroy session |

The TPM user could then use the session for further authorization sessions or terminate it in the ways that have been described above in TPM_OIAP. Note that termination of the OSAP session causes the TPM to destroy the shared secret.

*End of informative comment.*

### 5.2.5 TPM_OSAP

Start of informative comment:

The TPM_OSAP command creates the authorization handle, the shared secret and generates nonceEven and nonceEvenOSAP.

End of informative comment.

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_OSAP. |
| 4 | 2 | | | TCPA_ENTITY_TYPE | entityType | The type of entity in use |
| 5 | 4 | | | UINT32 | entityValue | The selection value based on entityType, e.g. a keyHandle # |
| 6 | 20 | | | TCPA_NONCE | nonceOddOSAP | The nonce generated by the caller associated with the shared secret. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | TCPA_AUTHHANDLE | authHandle | Handle that TPM creates that points to the authorization state. |
| 5 | 20 | | | TCPA_NONCE | nonceEven | Nonce generated by TPM and associated with session. |
| 6 | 20 | | | TCPA_NONCE | nonceEvenOSAP | Nonce generated by TPM and associated with shared secret. |

**Actions**

1. The TPM_OSAP command allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle, nonceEven and nonceEvenOSAP.
2. The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. The TPM_OSAP allows the binding of an authorization to a specific entity. This allows the caller to continue to send in authorization data for each command but not have to request the information or cache the actual authorization data.
4. Internally the TPM will do the following:
    a. TPM receives command.

b. TPM generates new handle and reserves space to save protocol identification, shared secret, both nonces and any other information the TPM needs to manage the session.

c. TPM generates nonces nonceEven and nonceEvenOSAP.

d. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC calculation is the secret authorization data assigned to the key handle identified by entityValue. The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved in the authorization area associated with authHandle

**Descriptions**

**entityType = TCPA_ET_KEYHANDLE**

The entity to authorize is a key held in the TPM. entityValue contains the keyHandle that holds the key.

**entityType = TCPA_ET_OWNER**

This value indicates that the entity is the TPM owner. entityValue is ignored.

**entityType = TCPA_ET_SRK**

The entity to authorize is the SRK. entityValue is ignored.

**Usage**

On each subsequent use of the OSAP session the TPM MUST generate a new nonce value.

The TPM MUST ensure that OS-AP shared secret is only available while the OS-AP session is valid.

**Termination**

The session MUST terminate upon any of the following conditions:

- The entity is unloaded.
- The entity has a change authorization performed on it.
- The session is used in a TPM_ChangeAuth command.
- The command that uses the session returns an error.

### 5.2.6 Authorization using an OS-AP session

*Start of informative comment:*

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OS-AP protocol.

Many commands use OS-AP authorization. The following description is therefore necessarily abstract.

*End of informative comment.*

**Actions**

On reception of a command with ordinal C1 that uses an authorization session, the TPM SHALL perform the following actions:

1. The TPM MUST have been able to retrieve the shared secret (Shared, say) of the target entity when the authorization session was established with TPM_OSAP. The entity and its secret must have been previously loaded into the TPM.
2. The TPM MUST verify that the authorization handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM_AUTHFAIL.
3. The TPM MUST calculate the HMAC (HM1, say) of the command parameters according to section 4.4.2
4. The TPM SHALL compare HM1 to the authorization value received in the command. If they are different, the TPM returns the error code TPM_AUTHFAIL. Otherwise, the TPM executes command C1 which produces an output (O, say) that requires authentication and uses a particular return code (RC, say).
5. The TPM SHALL generate the latest version of the even nonce (nonceEven).
6. The TPM MUST calculate the HMAC (HM2) of the return parameters according to section 4.4.2
7. The TPM returns HM2 in the parameter list.
8. The TPM SHALL retrieve the continue flag from the received command. If the flag is FALSE, the TPM SHALL terminate the session and destroy the thread associated with handle H.

If the shared secret was used to provide confidentiality for data in the received command, the TPM SHALL terminate the session and destroy the thread associated with handle H.

Each time that access to an entity (key) is authorized using OSAP, the TPM MUST ensure that the OSAP shared secret is that derived from the entity using TPM_OSAP.

## 5.3 TPM_Terminate_Handle

*Start of informative comment:*

This allows the TPM manager to clear out information in a session handle.

The TPM may maintain the authorization session even though a key attached to it has been unloaded or the authorization session itself has been unloaded in some way. When a command is executed that requires this session, it is the responsibility of the external software to load both the entity and the authorization session information prior to command execution.

*End of informative comment.*

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Terminate_Handle. |
| 4 | 4 | | | TCPA_AUTHHANDLE | handle | The handle to terminate |

Outgoing Operands and Sizes

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Descriptions**

A TPM SHALL unilaterally perform the actions of TPM_Terminate_Handle upon detection of the following events:

- Completion of a received command whose authorization "continueUse" flag is FALSE.
- Completion of a received command when a shared secret derived from the authorization session was exclusive-or'ed with data (to provide confidentiality for that data). This occurs during execution of a TPM_ChangeAuth command, for example.
- When the associated entity is destroyed (in the case of TPM Owner or SRK, for example)
- Upon execution of TPM_Init
- When the command returns an error. This is due to the fact that when returning an error the TPM does not send back nonceEven. There is no way to maintain the rolling nonces, hence the TPM MUST terminate the authorization session.
- Failure of an authorization check belonging to that authorization session.

**Actions**

The TPM SHALL terminate the session and destroy all data associated with the session indicated.

## 5.4 ADIP – Creating a New Entity

*Start of informative comment*

The creation of the authorization data is the responsibility of the entity owner. He or she may use whatever process he or she wishes. The transmission of the authorization data from the owner to the TPM requires confidentiality and integrity. The encryption of the authorization data meets these requirements. The confidentiality and integrity requirements assume the insertion of the authorization data occurs over a network. While local insertions of the data would not require these measures, the protocol is established to be consistent with both local and remote insertions.

When the requestor is sending the authorization data to the TPM, the command to load the data requires the authorization of the entity owner. For example, to create a new TPM ID and set is authorization data requires the authorization data of the TPM Owner.

The confidentiality of the transmission comes from the encryption of the authorization data, and the integrity comes from the ability of the owner to verify that the authorization is being sent to a TPM and that only a specific TPM can decrypt the data.

The mechanism uses the following features of the TPM: OS-AP and HMAC.

- The creation of a new entity requires the authorization of the entity owner. When the requestor starts the creation process, the creator must use OS-AP.
- The creator builds an encryption key using a SHA-1 hash of the shared secret from the OS-AP mechanism and the nonce (authLastNonceEven) returned by the TPM from the TPM_OSAP command.
- The creator encrypts the new authorization data using the key from the previous step as a one-time pad with XOR and then sends this encrypted data along with the creation request to the TPM.
- The TPM decrypts the authorization data using the OS-AP shared secret and authLastNonceEven creates the new entity.
- The TPM includes the sends the reply back to the creator using the new authorization data as the secret value of the HMAC.

The creator believes that the OS-AP creates a shared secret known only to the creator and the TPM. The TPM believes that the creator is the entity owner by their knowledge of the parent entity authorization data. The creator believes that the process completed correctly and that the authorization data is correct because the HMAC will only verify with the OS-AP secret.

The ADIP allows for the creation of new entities and the secure insertion of the new entity authorization data. The transmission of the new authorization data uses encryption with the key being a shared secret of an OS-AP session.

The OS-AP session must be created using the owner of the new entity.

In the following example, we want to send the previously described command TPM_EXAMPLE to create a new entity. In the example, we assume there is a third input parameter newAuth, and that one of the input parameters is named parentHandle to reference the parent for the new entity (TPM Owner in some circumstances such as the SRK and its children, otherwise a key).

| Caller | On the wire | Dir | TPM |
|---|---|---|---|
| Send TPM_OSAP | TPM_OSAP<br>parentHandle<br>nonceOddOSAP | → | ○ Create session & authHandle<br>○ Generate authLastNonceEven<br>○ Save authLastonceEven with authHandle<br>○ Generate nonceEvenOSAP<br>○ Generate sharedSecret = HMAC(parent.usageAuth, nonceEvenOSAP, nonceOddOSAP)<br>○ Save parentHandle, sharedSecret with authHandle |
| ○ Save authHandle, authLastNonceEven<br>○ Generate sharedSecret = HMAC(parent.usageAuth, nonceEvenOSAP, nonceOddOSAP)<br>○ Save sharedSecret | authHandle,<br>authLastNonceEven<br>nonceEvenOSAP | ← | Returns |
| ○ Generate nonceOdd & save with authHandle<br>○ Compute input parameter newAuth = XOR( entityAuthData, SHA1(sharedSecret, authLastNonceEven) )<br>○ Compute inAuth = HMAC (sharedSecret, inParamDigest, inAuthSetupParams) | | | |
| Send TPM_Example | tag<br>paramSize<br>ordinal<br>inArgOne<br>inArgTwo<br>newAuth<br>authHandle<br>nonceOdd<br>continueAuthSession<br>inAuth | → | ○ Verify authHandle points to a valid session, mismatch returns TPM_AUTHFAIL<br>○ Retrieve authLastNonceEven from internal session storage<br>○ HM = HMAC (sharedSecret, inParamDigest, inAuthSetupParams)<br>○ Compare HM to inAuth. If they do not compare return with TPM_AUTHFAIL<br>○ Compute entityAuthData = XOR( newAuth, SHA1(sharedSecret, authLastNonceEven))<br>○ Execute TPM_Example, create entity and build returnCode<br>○ Generate nonceEven to replace authLastNonceEven in session<br>○ Set resAuth = HMAC(sharedSecret, outParamDigest, outAuthSetupParams) |

| | | | |
|---|---|---|---|
| • Save nonceEven<br>• HM = HMAC( sharedSecret, outParamDigest, outAuthSetupParams)<br>• Compare HM to resAuth. This verifies returnCode and output parameters. | tag, paramSize, returnCode, outArgOne, nonceEven, continueAuthSession, resAuth | ← | • Return output parameters.<br>• Destroy auth session associated with authHandle |

End of informative comment

The TPM MUST enable ADIP by using the OS-AP. The TPM MUST encrypt the authorization data for the new entity by performing an XOR using the shared secret created by the OS-AP.

The TPM MUST destroy the OS-AP session whenever a new entity is created.

## 5.5 ADCP - Changing Authorization Data

***Start of informative comment:***

All entities from the Owner to the SRK to individual keys and data blobs have authorization data. This data may need to change at some point in time after the entity creation. The ADCP allows the entity owner to change the authorization data. The entity owner of a wrapped key is the owner of the parent key.

A requirement is that the owner must remember the old authorization data. The only mechanism to change the authorization data when the entity owner forgets the current value is to delete the entity and then recreate it.

To protect the data from exposure to eavesdroppers or other attackers, the authorization data uses the same encryption mechanism in use during the ADIP.

Changing authorization data requires opening two authentication handles. The first handle authenticates the entity owner (or parent) and the right to load the entity. This first handle is an OS-AP and supplies the data to encrypt the new authorization data according to the ADIP protocol. The second handle can be either an OI-AP or an OS-AP, it authorizes access to the entity for which the authorization data is to be changed.

The authorization data in use to generate the OS-AP shared secret must be the authorization data of the parent of the entity to which the change will be made.

When changing the authorization data for the SRK, the first handle OS-AP must be setup using the TPM Owner authorization data. This is because the SRK does not have a parent, per se.

If the SRKAuth data is known to userA and userB, userA can snoop on userB while userB is changing the authorisation for a child of the SRK, and deduce the child's newAuth. Therefore, if SRKAuth is a well known value, TPM_ChangeAuthAsymStart and TPM_ChangeAuthAsymFinish are preferred over TPM_ChangeAuth when changing authorisation for children of the SRK.

This applies to all children of the SRK, including TPM identities.

***End of informative comment.***

Changing authorization data for the TPM SHALL require authorization of the current TPM Owner.

Changing authorization data for the SRK SHALL require authorization of the TPM Owner.

If SRKAuth is a well known value, TPM_ChangeAuth SHOULD NOT be used to change the authorisation value of a child of the SRK, including the TPM identities.

All other entities SHALL require authorization of the parent entity.

## 5.6 Changing authorization values

*Start of informative comment.*

Changing authorization comes in two flavors one to handle blobs with authorization and one to handle the authorization for the TPM Owner and SRK.

Functionally these two commands perform the same operation and operate on the same fields the only difference lies in who authorizes the operation and where the data comes from.

*End of informative comment.*

### 5.6.1 TPM_ChangeAuth

*Start of informative comment.*

The TPM_ChangeAuth command allows the owner of an entity to change the authorization data for the entity.

TPM_ChangeAuth requires the encryption of one parameter ("NewAuth"). For the sake of uniformity with other commands that require the encryption of more than one parameter, the string used for XOR encryption is generated by concatenating the evenNonce (created during the OSAP session) with the session shared secret and then hashing the result.

The parameter list to this command must always include two authorization sessions, regardless of the state of authDataUsage for the respective keys.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorizations for the entity pointed to by parentHandle and inData.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_ChangeAuth |
| 4 | *4* | | | TCPA_KEY_HANDLE | parentHandle | Handle of the parent key to the entity. |
| 5 | *2* | 2s | *2* | TCPA_PROTOCOL_ID | protocolID | The protocol in use. |
| 6 | *20* | 3s | *20* | TCPA_ENCAUTH | newAuth | The encrypted new authorization data for the entity. The encryption key is the shared secret from the OS-AP protocol. |
| 7 | *2* | 4s | *2* | TCPA_ENTITY_TYPE | entityType | The type of entity to be modified |
| 8 | *4* | 5s | *4* | UINT32 | encDataSize | The size of the encData parameter |
| 9 | <> | 6s | <> | BYTE[ ] | encData | The encrypted entity that is to be modified. |
| 10 | *4* | | | TCPA_AUTHHANDLE | parentAuthHandle | The authorization handle used for the parent key. |
| | | 2H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | *20* | 3H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | parentAuthHandle |
| 12 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Ignored, parentAuthHandle is always terminated. |
| 13 | 20 | | | TCPA_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |
| 14 | 4 | | | TCPA_AUTHHANDLE | entityAuthHandle | The authorization handle used for the encrypted entity. The session type MUST be OIAP |
| | | 2 H2 | 20 | TCPA_NONCE | entitylastNonceEven | Even nonce previously generated by TPM |
| 15 | 20 | 3 H2 | 20 | TCPA_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 16 | 1 | 4 H2 | 1 | BOOL | continueEntitySession | Ignored, entityAuthHandle is always terminated. |
| 17 | 20 | | | TCPA_AUTHDATA | entityAuth | The authorization digest for the inputs and encrypted entity. HMAC key: entity.usageAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_ChangeAuth |
| 4 | 4 | 3s | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 5 | <> | 4s | <> | BYTE[ ] | outData | The modified, encrypted entity. |
| 6 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 7 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed value of FALSE |
| 8 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters and parentHandle. HMAC key: parentKey.usageAuth. |
| 9 | 20 | 2 H2 | 20 | TCPA_NONCE | entityNonceEven | Even nonce newly generated by TPM to cover entity |
| | | 3 H2 | 20 | TCPA_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 10 | 1 | 4 H2 | 1 | BOOL | entityContinueAuthSession | Continue use flag, fixed value of FALSE |
| 11 | 20 | | | TCPA_AUTHDATA | entityAuth | The authorization digest for the returned parameters and entity. HMAC key: newly changed entity.usageAuth. |

**Descriptions**

A TPM MUST support the TPM_PID_ADCP protocol.

**TPM_PID_ADCP protocol descriptions**

The parentAuthHandle session type MUST be TCPA_PID_OSAP.

**TPM_PID_ADCP protocol actions**

1. Verify that entityType is one of TCPA_ET_DATA, TCPA_ET_KEY and return the error TCPA_WRONG_ENTITYTYPE if not.
2. The encData field MUST be the encData field from either the TCPA_STORED_DATA or TCPA_KEY structures.
3. Create s1 string by concatenating (parentAuthHandle -> shared secret || authLastNonceEven)
4. Create x1 by performing a SHA1 hash of s1
5. Create decryptAuth by XOR of x1 and newAuth.
6. parentAuthHandle MUST be built using the parent entity's authorization data.
7. The TPM MUST validate the command using the authorization data in the parentAuth parameter. The parentRef parameter provides the identification of the parent.
8. After parameter validation the TPM creates b1 by decrypting inData using the key pointed to by parentHandle.
9. The TPM MUST validate that b1 is a valid TCPA structure by verifying that the command has been authorized to use the blob. This checks that 20B of the decrypted blob have the proper value, and provides statistical proof that the blob was correctly decrypted.
10. The TPM replaces the authorization data for b1 with decryptAuth created above.
11. The TPM encrypts b1 using the appropriate mechanism for the type using the parentKeyHandle to provide the key information.
12. The new blob is returned in outData when appropriate.
13. The TPM MUST enforce the destruction of both the parentAuthHandle and entityAuthHandle sessions.

### 5.6.2 TPM_ChangeAuthOwner

*Start of informative comment:*

The TPM_ChangeAuthOwner command allows the owner of an entity to change the authorization data for the TPM Owner or the SRK.

This command requires authorization from the current TPM Owner to execute.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorizations from the TPM Owner

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | $1_S$ | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthOwner |
| 4 | *2* | $2_S$ | *2* | TCPA_PROTOCOL_ID | protocolID | The protocol in use. |
| 5 | *20* | $3_S$ | *20* | TCPA_ENCAUTH | newAuth | The encrypted new authorization data for the entity. The encryption key is the shared secret from the OS-AP protocol. |
| 6 | *2* | $4_S$ | *2* | TCPA_ENTITY_TYPE | entityType | The type of entity to be modified |
| 7 | *4* | | | TCPA_AUTHHANDLE | ownerAuthHandle | The authorization handle used for the TPM Owner. |
| | | $2_{H1}$ | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | *20* | $3_{H1}$ | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 9 | *1* | $4_{H1}$ | *1* | BOOL | continueAuthSession | Continue use flag the TPM ignores this value |
| 10 | *20* | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and ownerHandle. HMAC key: tpmOwnerAuth. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal TPM_ORD_ChangeAuthOwner |
| 4 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed value of FALSE |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters and ownerHandle. HMAC key: tpmOwnerAuth. This is the new tpmOwnerAuth value if this command changed that value. |

**Descriptions**

A TPM MUST support the TPM_PID_ADCP protocol.

In this capability, the SRK cannot be accessed as entityType TCPA_ET_KEY, since the SRK is not wrapped by a parent key.

**TPM_PID_ADCP protocol descriptions**

The ownerAuthHandle session type MUST be TCPA_PID_OSAP.

**TPM_PID_ADCP protocol actions**

1. Verify that entityType is either TCPA_ET_OWNER or TCPA_ET_SRK, and return the error TCPA_WRONG_ENTITYTYPE if not.
2. The ownerAuthHandle -> entityType MUST be TCPA_ET_OWNER.
3. Create s1 string by concatenating (ownerAuthHandle -> shared secret || authLastNonceEven)
4. Create x1 by performing a SHA1 hash of s1
5. Create decryptAuth by XOR of x1 and newAuth.
6. The TPM MUST enforce the destruction of the ownerAuthHandle session upon completion of this command (successful or unsuccessful). This includes setting continueAuthSession to FALSE
7. Set the authorization data for the indicated entity to decryptAuth

## 5.7 Asymmetric Authorization Change Protocol

*Start of informative comment:*

This asymmetric change protocol allows the entity owner to change entity authorization, under the parent's execution authorization, to a value of which the parent has no knowledge.

In contrast, the TPM_ChangeAuth command uses the parent entity authorization data to create the shared secret that encrypts the new authorization data for an entity. This creates a situation where the parent entity ALWAYS knows the authorization data for entities in the tree below the parent. There may be instances where this knowledge is not a good policy.

This asymmetric change process requires two commands and the use of an authorization session.

*End of informative comment.*

Changing authorization data for the SRK SHALL involve authorization by the TPM Owner.

If SRKAuth is a well known value,

TPM_ChangeAuthAsymStart and TPM_ChangeAuthAsymFinish SHOULD be used to change the authorisation value of a child of the SRK, including the TPM identities.

All other entities SHALL involve authorization of the parent entity.

### 5.7.1 TPM_ChangeAuthAsymStart

*Start of informative comment:*

The TPM_ChangeAuthAsymStart starts the process of changing authorization for an entity. It sets up an OIAP session that must be retained for use by its twin TPM_ChangeAuthAsymFinish command.

TPM_ChangeAuthAsymStart creates a temporary asymmetric public key "tempkey" to provide confidentiality for new authorization data to be sent to the TPM. TPM_ChangeAuthAsymStart certifies that tempkey was generated by a genuine TPM, by generating a certifyInfo structure that is signed by a TPM identity. The owner of that TPM identity must cooperate to produce this command, because TPM_ChangeAuthAsymStart requires authorization to use that identity.

It is envisaged that tempkey and certifyInfo are given to the owner of the entity whose authorization is to be changed. That owner uses certifyInfo and a TPM_IDENTITY_CREDENTIAL to verify that tempkey was generated by a genuine TPM. This is done by verifying the TPM_IDENTITY_CREDENTIAL using the public key of a CA, verifying the signature on the certifyInfo structure with the public key of the identity in TPM_IDENTITY_CREDENTIAL, and verifying tempkey by comparing its digest with the value inside certifyInfo. The owner uses tempkey to encrypt the desired new authorization data and inserts that encrypted data in a TPM_ChangeAuthAsymFinish command, in the knowledge that only a TPM with a specific identity can interpret the new authorization data.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization for the identity in idHandle.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymStart. |
| 4 | *4* | | | TCPA_KEY_HANDLE | idHandle | The keyHandle identifier of a loaded identity ID key |
| 5 | *20* | 2s | *20* | TCPA_NONCE | antiReplay | The nonce to be inserted into the certifyInfo structure |
| 6 | <> | 3s | <> | TCPA_KEY_PARMS | tempKey | Structure contains all parameters of ephemeral key. |
| 7 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for idHandle authorization. |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | *20* | | | TCPA_AUTHDATA | idAuth | The authorization digest for inputs and idHandle. HMAC key: idKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymStart |
| 7 | 95 | 3s | 95 | TCPA_CERTIFY_INFO | certifyInfo | The certifyInfo structure that is to be signed. |
| 8 | 4 | 4s | 4 | UINT32 | sigSize | The used size of the output area for the signature |
| 9 | <> | 5s | <> | BYTE[ ] | sig | The signature of the certifyInfo parameter. |
| 10 | 4 | 6s | 4 | TCPA_KEY_HANDLE | ephHandle | The keyHandle identifier to be used by ChangeAuthAsymFinish for the ephemeral key |
| 11 | <> | 7s | <> | TCPA_KEY | tempKey | Structure containing all parameters and public part of ephemeral key. TCPA_KEY.encSize is set to 0. |
| 12 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 13 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 14 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: idKey.usageAuth. |

### Actions

1. The TPM SHALL verify the authorization to use the TPM identity key held in idHandle. The TPM MUST verify that the key is a TPM identity key.
2. The TPM SHALL validate the algorithm parameters for the key to create from the tempKey parameter.
    a. Recommended key type is RSA
    b. Minimum RSA key size MUST is 512 bits, recommended RSA key size is 1024
    c. For other key types the minimum key size strength MUST be comparable to RSA 512
3. The TPM SHALL create a new key (k1) in accordance with the algorithm parameter. The newly created key is pointed to by ephHandle.
4. The TPM SHALL fill in all fields in tempKey using k1 for the information. The TCPA_KEY -> encSize MUST be 0.
5. The TPM SHALL fill in certifyInfo using k1 for the information. The certifyInfo -> data field is supplied by the antiReplay.
6. The TPM then signs the certifyInfo parameter using the key pointed to by idHandle. The resulting signed blob is returned in sig parameter

**Field Descriptions for certifyInfo parameter**

| Type | Name | Description |
|---|---|---|
| TCPA_VERSION | `Version` | TCPA version structure; section 4.5. |
| keyFlags | `Redirection` | This SHALL be set to FALSE |
| | `Migratable` | This SHALL be set to FALSE |
| | `Volatile` | This SHALL be set to TRUE |
| TCPA_AUTH_DATA _USAGE | `authDataUsage` | This SHALL be set to TPM_AUTH_NEVER |
| TCPA_KEY_USAGE | `KeyUsage` | This SHALL be set to TPM_KEY_AUTHCHANGE |
| UINT32 | `PCRInfoSize` | This SHALL be set to 0 |
| TCPA_DIGEST | `pubDigest` | This SHALL be the hash of the public key being certified. |
| TCPA_NONCE | `Data` | This SHALL be set to antiReplay |
| TCPA_KEY_PARMS | `info` | This specifies the type of key and its parameters. |
| BOOL | `parentPCRStatus` | This SHALL be set to FALSE. |

### 5.7.2 TPM_ChangeAuthAsymFinish

*Start of informative comment:*

The TPM_ChangeAuth command allows the owner of an entity to change the authorization data for the entity.

The command requires the cooperation of the owner of the parent of the entity, since authorization must be provided to use that parent entity. The command requires knowledge of the existing authorization information and passes the new authorization information. The newAuthLink parameter proves knowledge of existing authorization information and new authorization information. The new authorization information "encNewAuth" is encrypted using the "tempKey" variable obtained via TPM_ChangeAuthAsymStart.

A parent therefore retains control over a change in the authorization of a child, but is prevented from knowing the new authorization data for that child.

The changeProof parameter provides a proof that the new authorization value was properly inserted into the entity. The inclusion of a nonce from the TPM provides an entropy source in the case where the authorization value may be in itself be a low entropy value (hash of a password etc).

*End of informative comment.*

**Type**

TCPA protected capability; caller must provide authorizations for the entity pointed to by parentRef and blob.

Incoming Operands and Sizes

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymFinish |
| 4 | *4* | | | TCPA_KEY_HANDLE | parentHandle | The keyHandle of the parent key for the input data |
| 5 | *4* | | | TCPA_KEY_HANDLE | ephHandle | The keyHandle identifier for the ephemeral key |
| 6 | *2* | 3s | *2* | TCPA_ENTITY_TYPE | entityType | The type of entity to be modified |
| 7 | *20* | 4s | *20* | TCPA_HMAC | newAuthLink | HMAC calculation that links the old and new authorization values together |
| 8 | *4* | 5s | *4* | UINT32 | newAuthSize | Size of encNewAuth |
| 9 | <> | 6s | <> | BYTE[ ] | encNewAuth | New authorization data encrypted with ephemeral key. |
| 10 | *4* | 7s | *4* | UINT32 | encDataSize | The size of the inData parameter |
| 11 | <> | 8s | <> | BYTE[ ] | encData | The encrypted entity that is to be modified. |
| 12 | *4* | | | TCPA_AUTHHANDLE | authHandle | Authorization for parent key. |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 13 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 14 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 15 | *20* | | | TCPA_AUTHDATA | privAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymFinish |
| 4 | 4 | 3s | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 5 | <> | 4s | <> | BYTE[ ] | outData | The modified, encrypted entity. |
| 6 | 20 | 5s | 20 | TCPA_NONCE | saltNonce | A nonce value from the TPM RNG to add entropy to the changeProof value |
| 7 | <> | 6s | <> | TCPA_DIGEST | changeProof | Proof that authorization data has changed. |
| 8 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 10 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |

**Description**

If the parentHandle points to the SRK then the HMAC key MUST be built using the TPM Owner authorization.

**Actions**

1. The TPM SHALL validate that the authHandle parameter authorizes use of the key in parentHandle.
2. The encData field MUST be the encData field from TCPA_STORED_DATA or TCPA_KEY.
3. The TPM SHALL create e1 by decrypting the entity held in the encData parameter.
4. The TPM SHALL create a1 by decrypting encNewAuth using the authHandle -> TPM_KEY_AUTHCHANGE private key. a1 is a structure of type TCPA_CHANGEAUTH_VALIDATE.
5. The TPM SHALL create b1 by performing the following HMAC calculation: b1 = HMAC (a1 -> newAuthSecret). The secret for this calculation is encData -> currentAuth. This means that b1 is a value built from the current authorization value (encData -> currentAuth) and the new authorization value (a1 -> newAuthSecret).
6. The TPM SHALL compare b1 with newAuthLink. The TPM SHALL indicate a failure if the values do not match.
7. The TPM SHALL replace e1 -> authData with a1 -> newAuthSecret
8. The TPM SHALL encrypt e1 using the appropriate functions for the entity type. The key to encrypt with is parentHandle.
9. The TPM SHALL create saltNonce by taking the next 20 bytes from the TPM RNG.
10. The TPM SHALL create changeProof a HMAC of (saltNonce concatenated with a1 -> n1) using a1 -> newAuthSecret as the HMAC secret.
11. The TPM MUST destroy the TPM_KEY_AUTHCHANGE key associated with the authorization session.

## 5.8 Authorization Data

*Start of informative comment:*

The authorization data is a 160-bit field that the TPM stores in a "shielded location", which is an area where data is protected against interference and prying, independent of its form. The Owner has a copy of the data and protects the data using whatever mechanism the Owner wishes to use. The authorization data is a shared secret between the TPM and the Owner of the entity. There are no requirements as to what the 160 bit of data are. The assumption is that the data is a SHA-1 hash of a password or other data, but the data can be anything.

There will be a separate piece of authorization data for each entity. There is no requirement that each authorization data blob must be unique.

The TPM treats the authorization data as shielded data, an approach that requires that only TPM-protected capabilities access the authorization data. A further requirement is that the only use of the authorization data within the TPM is in the authorization process. No other use is permissible.

The protection of the backup mechanism is a type of authorization.

*End of informative comment.*

The TPM MUST reserve 160 bits for the authorization data. The TPM treats the authorization data as a blob. The TPM MUST keep the authorization data in a shielded location.

The TPM MUST enforce that the only usage in the TPM of the authorization data is to perform authorizations.

## 5.9 Nonces

*Start of informative comment:*

All of the authorization protocols require nonces to prevent replay and man-in-the-middle attacks. To further strengthen the use of the nonces a rolling nonce paradigm requires the use of new nonces for each message and response.

The nonce values from the TPM must use the internal RNG. The nonce values from the requestor can use any source that provides information to the requestor. The highest value is obtained when the requestor also uses an RNG for the nonce values; however, there is no loss of security to the TPM if set values are in use. The requestor loses some protection when he or she (or it) uses set values.

In all descriptions of nonce usage in this section all odd nonce values come from the challenger; all even nonce values come from the TPM (0 is an even number for this definition).

The requestor is responsible for generating and sending the odd nonce value. The TPM may enforce that the odd nonce value changes for each request.

The TPM is responsible for the even nonce values. The TPM changes the value of the even nonce on each reply.

*End of informative comment.*

The requestor SHOULD provide a unique value in the odd nonce field of the authorization structure for each request. The TPM MAY enforce the uniqueness of values from the requestor.

The TPM MUST supply a new nonce value for each reply. The nonce value MUST come from the internal RNG. The TPM MUST enforce the validity of the returning nonce another command uses the authorization session.

## 5.10 Authorization Handle

*Start of informative comment:*

The TPM generates authorization handles to allow for the tracking of information regarding a specific authorization invocation.

The TPM saves information specific to the authorization, such as the nonce values, ephemeral secrets and type of authentication in use.

The TPM may create any internal representation of the handle that is appropriate for the TPM's design. The requestor always uses the handle in the authorization structure to indicate authorization structure in use.

The TPM must support a minimum of two concurrent authorization handles. The use of these handles is to allow the Owner to have an authorization active in addition to an active authorization for an entity.

To ensure garbage collection and the proper removal of security information, the requestor should terminate all handles. Termination of the handle uses the continue-use flag to indicate to the TPM that the handle should be terminated.

Termination of a handle instructs the TPM to perform garbage collection on all authorization data. Garbage collection includes the deletion of the ephemeral secret.

*End of informative comment.*

The TPM MUST support authorization handles. The TPM MUST support a minimum of two concurrent authorization handles.

The TPM MUST support authorization-handle termination. The termination includes secure deletion of all authorization session information.

## 5.11 TPM Ownership

*Start of informative comment:*

The Owner of the TPM has the right to perform special operations. The process of taking ownership is the procedure whereby the Owner inserts a shared secret into the TPM. For all future operations, knowledge of the shared secret is proof of Ownership. When the Owner wishes to perform one of the special operations then the Owner must use the authorization protocol to prove knowledge of the shared secret.

The TPM default state is to have no Owner.

The difficulty with Ownership is inserting the shared secret in a secure manner. A design consideration is that the taking of Ownership must be an operation that works securely over the network. The function must provide confidentiality and integrity to the messages sent to the TPM.

The function to insert the Owner must provide the following:

- Confidentiality. The shared secret (or authorization data) must remain confidential to all eavesdroppers that intercept any of the messages. The confidentiality comes from encrypting the shared secret using the TPM PUBEK. The Owner trusts that only the TPM has the PRIVEK that can decrypt the shared secret.
- Integrity. The TPM and the Owner must be able to determine the integrity of messages and responses to the function. The integrity checking does not have to occur at the instant of receiving a message. The Owner validates the integrity of the messages using the HMAC construct.
- Remoteness – the function must allow the Owner to take control across a network.
- Verifiability. The function allows the Owner to verify that he or she has truly taken control. The Owner verifies that the secret was successfully installed by verifying the HMAC response. Additional verification can occur by attempting to establish a Owner session.

The TPM_TakeOwnership function inserts the Owner authorization data and creates a new Storage Root Key (SRK). The TPM_TakeOwnership function fails if there is already an Owner set for the TPM.

After inserting the authorization data, the TPM_TakeOwnership function creates the SRK. The SRK (like any other key) can be linked to a PCR.

To validate that the operation completes successfully, the TPM HMACs the response to the TPM_TakeOwnership function.

*End of informative comment.*

The TPM MUST ship with no Owner installed. The TPM MUST use the ownership-control protocol.

### 5.11.1 TPM_TakeOwnership

**Type**

TCPA protected capability; user must encrypt the values using the PUBEK.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_TakeOwnership |
| 4 | 2 | 2s | 2 | TCPA_PROTOCOL_ID | protocolID | The ownership protocol in use. |
| 5 | 4 | 3s | 4 | UINT32 | encOwnerAuthSize | The size of the encOwnerAuth field |
| 6 | <> | 4s | <> | BYTE[ ] | encOwnerAuth | The owner authorization data encrypted with PUBEK |
| 7 | 4 | 5s | 4 | UINT32 | encSrkAuthSize | The size of the encSrkAuth field |
| 8 | 256 | 6s | 256 | BYTE[ ] | encSrkAuth | The SRK authorization data encrypted with PUBEK |
| 9 | <> | 7s | <> | TCPA_KEY | srkParams | Structure containing all parameters of new SRK. pubKey.keyLength & encSize are both 0 |
| 10 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for this command |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 13 | 20 | | | TCPA_AUTHDATA | ownerAuth | Authorization digest for input params. HMAC key: the new ownerAuth value. See actions for validation operations |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_TakeOwnership |
| 4 | <> | 3s | <> | TCPA_KEY | srkPub | Structure containing all parameters of new SRK. srkPub.encData is set to 0. |
| 5 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: the new ownerAuth value |

**Actions**

The new owner MUST encrypt the Owner authorization data and the SRK authorization data using the PUBEK. The endorsement key pair MUST be an RSA key so the encryption algorithm in use to encrypt these secrets is RSA.

If the TPM has a current owner then the TPM upon receipt of this command SHALL return the error code TCPA_OWNER_SET.

If the TPM has no current owner then the TPM upon receipt of this command SHALL:

1. If no EK is present the TPM MUST return TCPA_NO_ENDORSEMENT
2. If TCPA_PERSISTENT_FLAGS -> ownership is FALSE, the TPM SHALL abandon the process of granting ownership and return the error TCPA_INSTALL_DISABLED
3. Verify that the authorization session is of type OI-AP.
4. Decrypt EncOwnerAuth using the PRIVEK to generate ProspectiveOwnerAuth.
5. Use the TCPA authorization protocol to verify that all input parameters tagged with AUTH have been sent by an entity that knows ProspectiveOwnerAuth.
6. Store ProspectiveOwnerAuth as the Owner's authorization data.
7. Generate a new SRK in accordance with the algorithm parameter. In version 1 of the specification, algorithm MUST indicate a 2048 bit RSA key.
8. Verify that srkParams->keyUsage is TPM_KEY_STORAGE. If it is not, return TCPA_BAD_PARAMETER" .
9. Verify that srkParams->keyFlags->migratable is FALSE. If it is not, return TCPA_BAD_PARAMETER"
10. Decrypt EncSrkAuth using the PRIVEK and store the result as the SRK's authorization data.
11. Obtain a TCPA_NONCE from the TPM's Random Number Generator and store it as TCPA_PERSISTENT_DATA -> tpmProof. tpmProof SHALL be stored in TCPA shielded locations, only.
12. Return the public part of the SRK to the caller.
13. Calculate an authenticated response using the new authorization data

# 6. Integrity Collection and Reporting

## 6.1 Introduction

*Start of informative comment:*

The TCPA Trusted Platform Support Services (TSS) provides mechanisms for cryptographically reporting the current hardware and software configuration of a computing device to local and remote Challengers. The TSS also provides a limited protected storage capability, which allows the Subsystem Owner to store an acceptable platform configuration, biometric data or other data that is available early in boot. System firmware or other software may use this storage capability to name Users qualified to log on, or acceptable boot configurations. TCPA specification does *not* define how this storage facility should be used.

The TSS also provides a facility whereby platform software or firmware may store secrets that are accessible only when the platform is in a defined configuration. This mechanism is known as *sealing*. The following sections describe and define the Trusted Platform Module (TPM)-protected operations that support integrity collection and reporting. The usage required in a TCPA-compliant PC platform is described in a separate document.

*End of informative comment.*

## 6.2 Platform Configuration Registers

### 6.2.1 Format and Properties

A Platform Configuration Register (PCR) consists of a 160-bit field that holds a cumulatively updated hash value and a 4-byte status field. The PCR data structure MUST be a TCPA-shielded location. PCRs SHOULD be in volatile storage. The PCRs MUST be set to 0 before first use. This specification does not mandate the internal storage format.

A TPM implementation MUST provide 16 or more independent PCRs. These PCRs are identified by index and MUST be numbered from 0 (that is, $PCR_0$ through $PCR_{15}$ are required for TCPA compliance). Vendors MAY implement more registers for general-purpose use. Extra registers MUST be numbered contiguously from 16 up to max – 1, where max is the maximum offered by the TPM.

The TCPA-protected capabilities that expose and modify the PCRs use a 32-bit index, indicating the maximum usable PCR index. However, TCPA reserves register indices $2^{30}$ and higher for later versions of the specification. A TPM implementation MUST NOT provide registers with indices greater than or equal to $2^{30}$. In this specification, the following terminology is used (although this internal format is not mandated).

### 6.2.2 Initialization

PCRs and the protected capabilities that operate upon them MAY NOT be used until power-on self-test (TPM POST) has completed. If TPM POST fails, the TPM_Extend operation will fail; and, of greater importance, the TPM_Quote operation and TPM_Seal operations that respectively report and examine the PCR contents MUST fail. At the successful completion of TPM POST, all PCRs MUST be set to 0. Additionally, the UINT32 flags MUST be set to zero.

### 6.2.3 Authorized PCRs

A TPM MUST provide one Data Integrity Register (DIR). Implementations MAY provide more. These registers MUST hold 160-bit values and MUST be held in TCPA-shielded locations. Further, these registers MUST be non-volatile (values are maintained during the power-off state). A TPM implementation need not provide the same number of DIRs as PCRs.

## 6.3 Operations Supporting Integrity Collection and Reporting

### 6.3.1 TPM_Extend

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Extend. |
| 4 | 4 | | | TCPA_PCRINDEX | pcrNum | The PCR to be updated. |
| 5 | 20 | | | TCPA_DIGEST | inDigest | The 160 bit value representing the event to be recorded. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 20 | | | TCPA_PCRVALUE | outDigest | The PCR value after execution of the command. |

**Descriptions**

TPM_Extend, TPM_SHA1CompleteExtend and TPM_Startup SHALL be the only commands that alter the value of any PCRs.

When TCPA_PERSISTENT_FLAG -> disable is TRUE, TPM_Extend SHALL update the target PCR but return zero instead of the new value of the PCR.

**Actions**

1. Create c1 by concatenating ($PCR_{index}$ TCPA_PCRVALUE || inDigest). This takes the current PCR value and concatenates the inDigest parameter.
2. Create h1 by performing a SHA1 digest of c1.
3. Store h1 as the new TCPA_PCRVALUE of $PCR_{index}$
4. **If TCPA_PERSISTENT_FLAG -> disable is TRUE**
    a. Set outDigest to 20 bytes of 0x00
5. Else
    a. Set outDigest to h1

### 6.3.2 TPM_PcrRead

Start of informative comment:

The TPM_PcrRead operation provides non-cryptographic reporting of the contents of a named PCR.

End of informative comment

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_PcrRead. |
| 4 | 4 | | | TCPA_PCRINDEX | pcrIndex | Index of the PCR to be read |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 20 | | | TCPA_PCRVALUE | outDigest | The current contents of the named PCR |

**Actions**

The TPM_PcrRead operation returns the current contents of the named register to the caller.

### 6.3.4 TPM_DirWriteAuth

*Start of informative comment:*

The TPM_DirWriteAuth operation provides write access to the Data Integrity Registers. DIRs are non-volatile memory registers held in a TCPA-shielded location. Owner authentication is required to authorize this action. Version 1 requires only one DIR. If the DIR named does not exist, the TPM_DirRead operation returns TCPA_BADINDEX.

*End of informative comment.*

**Type**

TCPA protected capability; the user must provide authorization from the TPM Owner to execute function.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DirWriteAuth. |
| 4 | 4 | 2s | 4 | TCPA_DIRINDEX | dirIndex | Index of the DIR |
| 5 | 20 | 3s | 20 | TCPA_DIRVALUE | newContents | New value to be stored in named DIR |
| 6 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for command. |
| | | 2H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DirWriteAuth |
| 4 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. Validate that authHandle contains a TPM Owner authorization to excute the TPM_DirWriteAuth command
2. Validate that dirIndex points to a valid DIR on this TPM
3. Write newContents into the DIR pointed to by dirIndex

### 6.3.5 TPM_DirRead

*Start of informative comment:*

The TPM_DirRead operation provides read access to the DIRs. No authentication is required to perform this action because typically no cryptographically useful authorization data is available early in boot. TSS implementors may choose to provide other means of authorizing this action. Version 1 requires only one DIR. If the DIR named does not exist, the TPM_DirRead operation returns TCPA_BADINDEX.

*End of informative comment.*

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_DirRead. |
| 4 | *4* | | | TCPA_DIRINDEX | dirIndex | Index of the DIR to be read |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | *20* | | | TCPA_DIRVALUE | dirContents | The current contents of the named DIR |

**Actions**

1. Validate that dirIndex points to a valid DIR on this TPM
2. Return the contents of the DIR in dirContents

# 7. Protected Storage

*Start of informative comment:*

This section introduces the processes by which a TPM may act as the portal to confidential data stored on arbitrary storage media.

A TPM is required to protect the keys that represent TPM identities, and keys that are released only when the computing environment of the associated platform has a particular state. Given this capability, it is a natural extension to enable a TPM to protect arbitrary data and arbitrary keys. Unfortunately, this approach requires a potentially unbounded amount of storage within a TPM. The TCPA specification therefore includes capabilities that enable a TPM to act as a portal to potentially unbounded amounts of confidential data outside the TPM.

Storing data outside the TPM has the additional advantages of enabling easier migration of confidential data from one platform to another and enabling recovery of confidential data in the event of platform failure. These protected-storage capabilities are designed to enable the TPM to operate as a slave device so as to avoid the cost/complexity associated with a master device in a computing platform. These capabilities also are designed to avoid the need for the TPM to manage the confidential data that is stored outside the TPM. These design goals impose constraints on the nature of the protected-storage capabilities.

The TCPA solution uses the TPM to generate 'blobs' of secret data. Unspecified capabilities outside the Subsystem manage protected storage and issue certificates or other indications about the purpose and usefulness of data/keys held in blobs. Those unspecified capabilities issue commands to the TPM that cause it to create blobs of data and to use and return the contents of such blobs. This unspecified functionality is the manager of protected storage and uses the TPM as a specialized co-processor. The protected-storage commands are chosen to prevent subversion of the data in protected storage. Hence a rogue management function can disrupt protected storage but cannot subvert it.

A stored secret could be any of the following:

- Arbitrary data or a key. If a secret is arbitrary data, it can be exported from the TPM, and the TPM will not perform operations using that data. If the secret is a key, it is available for use within the TPM and will never be exported from the TPM.
- An encryption (storage) key or a signing key. If a key is for encryption it must not be used for signing and visa versa. Encryption keys are used only to provide confidentiality for blobs. Signature keys are used for signing arbitrary data submitted by the entity authorized to use that key.
- The signature key of a TPM identity. Such a signature key will be used only for special signing operations.

A stored secret has the following attributes:

- It may be capable of migration to another platform or it may be non-migratable. Keys that are migratable cannot be considered unique to a particular platform. Non-migratable keys can be considered to be unique to a particular platform.
- It may be generated inside the TPM or externally loaded. Externally loaded keys cannot be stored as non-migratable keys, for obvious reasons.
- It may be bound to the TPM or bound to a sequence of integrity metrics. At times, data or a key is required to be bound to a particular platform. At other times, it is required to be bound to a particular computing environment within a platform.
- It may have access control. A secret may be open to all processes on a platform or it may not, with varying degrees of control in between.

Some of these attributes are partitioned as separate commands, while others are partitioned as flags within commands. All the commands cause the TPM to create a secret blob and return it to the caller. The inverse commands cause the TPM to import a blob. Sometimes the TPM will then return the contents of

the blob (data) to the caller, and sometimes the TPM loads the contents of the blob (a key) for use within the TPM.

In all cases, the TPM must already contain the key that will be used to either encrypt or decrypt the blob. This naturally leads to a tree of blobs, where intermediate nodes contain encryption (storage) keys that are used to encrypt/decrypt child nodes. The root of the tree is the "Storage Root Key" (SRK) which is generated inside the TPM and is non-migratable. Only leaf nodes can contain signing keys, because a TPM will refuse to use a signing key to encrypt/decrypt child nodes. A TPM also will refuse to use a migratable node as the parent of a non-migratable node. (This enables migration of the supposedly non-migratable node.) On the other hand, a non-migratable node could be the parent of a migratable node with no ill effects.

The commands executed by the TPM are as follows:

- TSS_Bind: External data is encrypted under a parent key. (TPM_UnBind decrypts the blob using the parent key and exports the data from the TPM.)
- TPM_Seal: External data is concatenated with a value of integrity metric sequence and encrypted under a parent key. (TPM_Unseal decrypts the blob using the parent key and exports the plaintext data if the current integrity metric sequence inside the TPM matches the value of integrity metric sequence inside the blob.) The sealer of the data may specify that no integrity metrics are required.
- TSS_WrapKey: An externally generated key is encrypted under a parent key. (TPM_LoadKey decrypts the target blob using the parent key and loads the target key inside the TPM for use by the TPM.)
- TSS_WrapKeyToPcr: An externally generated key is concatenated with a value of integrity metric sequence and encrypted under a parent key. (TPM_LoadKey decrypts the target blob using the parent key and loads the target key inside the TPM, for use by the TPM, if the current integrity metric sequence inside the TPM matches the value of integrity metric sequence inside the blob.)
- TPM_CreateWrapKey: A key is generated inside the TPM, concatenated with a value of integrity metric sequence, and encrypted under a parent key. (TPM_LoadKey decrypts the target blob using the parent key and loads the target key inside the TPM, for use by the TPM, if the current integrity metric sequence inside the TPM matches the value of integrity metric sequence inside the blob.)

When a blob is loaded into a TPM, the TPM distinguishes between a data-bearing blob and a key-bearing blob by inspecting the data structure inside the blob. Data-bearing blobs are constructed according to PKCS #1. Key-bearing blobs are constructed using a TCPA-defined format. Each blob containing a key includes the field KeyUsage, which indicates whether the key is to be used for encryption (storage) or signing.

| Command | Usage with keys | Comment |
|---|---|---|
| TSS_Bind | N/A | No key |
| TPM_Seal | N/A | No key |
| TSS_WrapKey | Migratable encrypt or sign | Externally loaded |
| TSS_WrapKeyToPcr | Migratable encrypt or sign | Externally loaded |
| TPM_CreateWrapKey | Any | |

TCPA-protected storage uses asymmetric cryptography exclusively. One reason is that asymmetric crypto is already required to support TPM identities, but asymmetric crypto is not specifically necessary for any function. Another reason is that (in many, but not all, cases) operations to construct blobs can be performed outside the TPM; only the recovery of information from blobs (using the private key) must be done inside a TPM. This is possible because it is frequently true that all the necessary data to construct a blob (including the public key) is available outside the TPM. One notable exception is the TPM_Seal command, which must be performed inside a TPM because it requires reliable access to the Platform Configuration Registers and/or TMPProof. Using asymmetric crypto for protected storage therefore reduces the complexity of a TPM.

Some other important characteristics of "protected storage" are:

- Whenever a blob is created, the TPM includes random data to guard against plaintext attacks.
- Whenever a CreateWrapKey command creates a new key within the TPM, the blob that is produced contains the private (signature) key and the TPM also exports the corresponding public (identity) key as plaintext.
- Whenever a WrapXX command loads a new key into the TPM, only the private key (and its RSA modulus) must be presented.
- Whenever the TPM_LoadKey command is asserted, the TPM imports a secret blob containing the private (signature) key and the TPM also imports the corresponding public (identity) key as plaintext. Active RSA keys inside the TPM are referenced by handle where loaded into the TPM. To minimize key management burden inside the TPM, it is assumed "key slot" management is performed outside the TPM.
- The integrity of the data from the TPM_UnBind command is not checked by the TPM. Hence applications should use an "out of band" mechanism for verifying data integrity, if such verification is necessary.

Each secret blob contains a field of 20 bytes that may be used for authorization data. For convenience, the authorization field is the same size as the output of the SHA-1 hash algorithm. The authorization field is merely stored inside a blob, and the protected-storage capabilities do not themselves interpret the field.

The authDataUsage field determines when authorization is required.

The integrity of data or keys recovered from blobs is ensured by an implicit, rather than explicit mechanism. Ordinarily, an integrity check is provided by appending a checksum to original plaintext data. After decryption, the checksum is recomputed and compared with the checksum in the recovered data. Such a checksum needs to be at least 16 bytes long so as to have the necessary statistical properties. In the case of recovered blobs, the first 20 bytes of authorization data are sufficient to determine with high probability that data has been successfully decrypted without error. If the decryption fails, or the encrypted data contains errors, it is unlikely that the authorization data in the recovered blob will match the submitted authorization data.

The TPM also can be commanded to provide evidence that a particular public key is associated with a non-migratable private key (which was generated by the TPM and has never been released outside the TPM). This is the TPM_CertifyKey. It enables a third party to use a public key to encrypt data that can be recovered only using a protected-storage command. It also enables a third party to have confidence that a signature key has been generated by the TPM and has never been released outside the TPM.

Migratory data may be copied to an arbitrary number of platforms, using the "migration" commands provided. Non-migratory data may be moved to another platform only with the cooperation of a third party (the manufacturer of the platform, or his representative), using the "maintenance" commands provided.

*End of informative comment*

## 7.1 Introduction

### 7.1.1 Characteristics

*Start of informative comment*

This section specifies how to use the TPM to provide secure storage for an unlimited number of private keys or other data. Basically, this is done through the RSA key technology built into the TPM to encrypt data and keys with a public key to which the TPM has access to its corresponding private key. The resulting encrypted file – which contains header information in addition to the data or key, is called a blob and cannot be any bigger than key size used to encrypt it. The specification also shows how this is done so that private keys generated on the TPM can be stored outside the TPM (encrypted) in a way that allows the TPM to use them later without ever exposing such keys in the clear outside the TPM.

Padding and speed requirements make the TPM a very inefficient and inappropriate vehicle to do any bulk encryption, but it can be used to securely store keys that would then be used by software to do bulk encryption. There are a number of usage modules that imply requirements on the function of the TPM, as follows:

- Signing with a private key by the TPM can be accomplished only by presentation of authorization data to the TPM that is associated with that private key. A private key generated by a third party can be linked to a specific TPM without exposing the private key to the Owner/User of the TPM, but only with the consent of the User of the TPM.
- It MUST be possible to prove a specific public key is associated with a private key known only to a TPM. It must be possible for the Owner of a key, with the cooperation of the Owner of the TPM to migrate a migratable key from one platform to another without giving up control of the Key to the TPM Owner.
- It must not be possible for the Owner of a key, even with the cooperation of the Owner of the TPM to migrate a non-migratable key from one platform to another. Since a key may be wrapped outside the TPM, it is necessary that non-migratable keys always be generated inside the TPM. It must not be possible for the Owner of a non-migratable asymmetric key, even with cooperation of the Owner of the TPM, to decrypt the contents of an encrypted bundle encrypted with that non-migratable asymmetric key.
- If a TPM is compromised, it must not compromise all TPMs.
- To facilitate application level exchange of symmetric keys, the symmetric keys are stored using PKCS#1

All this is generally accomplished as follows:

- Any data in protected storage is explicitly identified as migratable or non-migratable.
- Each TPM contains a SRK, generated by the TPM at the request of the Owner. Under that SRK are two trees: one dealing with migratable data and the other dealing with non-migratable data.
- The non-migratable tree is directly below the SRK. The migration tree is directly below a "migration root" key that is directly below the SRK. Each node in a tree provides confidentiality for the nodes immediately below it. Obviously, all intermediate nodes in the trees must be encryption keys. Nodes in the non-migratable tree must be generated by the TPM, otherwise non-migratable nodes could be exposed.

Finally, some observations:

- In the migration tree, only leaf nodes should be available for signing. This is because a signature node (used outside the TPM for signing) should never be used for encryption and hence cannot be used to encrypt other nodes. Hence it must be a leaf.
- Similarly, in a non-migration tree, only leaf nodes should be available for signing. Since non-migratable nodes must not be migrated, they must never appear outside the TPM after being installed in the TPM.
- Any non-leaf node in the non-migratable tree must be generated within the TPM and never exposed outside the TPM. Any key (and hence every non-migratable key) generated in a TPM must be a genuine key.
- Any migratable key can be migrated by anyone that owns any of its migratable ancestors. As a result, in order to be sure that a migratable key cannot be migrated by anyone but the owner of that key, the owner can always create the migratable key and store it with a non-migratable storage key, thus guaranteeing the user has unique authority to authorize migration of that key.

***End of informative comment***

### 7.1.2 Key Storage

The number of asymmetric keys that are storable via a TPM SHOULD be limited only by the volume of storage available to the platform.

The TPM SHALL ensure that the TCPA_PERSISTENT_FLAGS -> tmpProof field is only included on TPM internally generated non-migratable keys. The rationale is that the tmpProof field is confidential information and exposure of this information would lower the security of the system.

## 7.2 Mandatory Functions

***Start of informative comment:***

Every TSS MUST support these functions; some must be TPM, and all may be TPM. They are derived from three parameters:

1. Is the secret stored data or as a key?
2. Is the secret generated internally or externally?
3. Is the secret bound to just the platform or also to PCRs?

These parameters would ordinarily lead to eight functions, but because data is always assumed to be generated externally, they yield to just six functions, as follows:

1. Data, generated externally, bound to PCRs: TPM_Seal command (TPM-protected capability). Inverse command is TPM_Unseal
2. Data, generated externally, bound to platform: TSS_Bind command (TSS). Inverse command is TPM_UnBind
3. Key, generated internally, bound to platform, bound to PCRs: TPM_CreateWrapKey command (TPM-protected capability). Inverse command is TPM_LoadKey
4. Key, generated externally, bound to PCRs: TSS_WrapKeyToPcr (TSS). Inverse command is TPM_LoadKey
5. Key, generated externally, bound to platform: TSS_WrapKey command (TSS). Inverse command is TPM_LoadKey

***End of informative comment.***

### 7.2.1 TPM_Seal

*Start of informative comment:*

The SEAL operation allows software to explicitly state the future "trusted" configuration that the platform must be in for the secret to be revealed. The SEAL operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual TPM.

If the UNSEAL operation succeeds, proof of the platform configuration that was in effect when the SEAL operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALed secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALed, and the proof may be of no interest. On the other hand, if the SEALed secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALed. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALed. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that *any* SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, *and* the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the *past* SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

TPM_Seal requires the encryption of one parameter ("Secret"). For the sake of uniformity with other commands that require the encryption of more than one parameter, the string used for XOR encryption is generated by concatenating a nonce (created during the OSAP session) with the session shared secret and then hashing the result.

*End of informative comment.*

**Type**

TPM function; user must provide authorization to use the key pointed to by keyHandle.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Seal. |
| 4 | *4* | | | TCPA_KEY_HANDLE | keyHandle | Handle of a loaded key that can perform seal operations. |
| 5 | *20* | 2s | *20* | TCPA_ENCAUTH | encAuth | The encrypted authorization data for the sealed data. The encryption key is the shared secret from the OS-AP protocol. |
| 6 | *4* | 3s | *4* | UINT32 | pcrInfoSize | The size of the pcrInfo parameter. If 0 there are no PCR registers in use |

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | <> | 4s | <> | TCPA_PCR_INFO | pcrInfo | The PCR selection information |
| 8 | 4 | 5s | 4 | UINT32 | inDataSize | The size of the inData parameter |
| 9 | <> | 6s | <> | BYTE[ ] | inData | The data to be sealed to the platform and any specified PCRs |
| 10 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization. Must be an OS_AP session for this command. |
| | | 2H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | 20 | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Ignored |
| 13 | 20 | | | TCPA_AUTHDATA | pubAuth | The authorization digest for inputs and keyHandle. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Seal. |
| 4 | <> | 3s | 4 | TCPA_STORED_DATA | sealedData | Encrypted, integrity-protected data object that is the result of the TPM_Seal operation. |
| 5 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed value of FALSE |
| 7 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth. |

### Descriptions

The string used for XOR encryption of the command variable named encAuth SHALL be the digest created by concatenating the shared session secret with the even numbered hash (generated by the TPM) and hashing the concatenated value.

TPM_Seal is used to encrypt private objects that can only be decrypted using TPM_Unseal.

### Actions

1. If the inDataSize is 0 the TPM returns TCPA_BAD_PARAMETER
2. If the keyUsage field of the key indicated by keyHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TCPA_INVALID_KEYUSAGE.
3. If the keyHandle points to a migratable key then the TPM MUST return the error code TCPA_INVALID_KEY_USAGE.

4. The TPM_Seal command MUST fill in a TPM_STORED_DATA structure. This structure includes a properly filled in and encrypted TCPA_SEALED_DATA structure. The encryption key for the operation is the key pointed to by the keyHandle parameter.
5. The TPM MUST set the TPM_STORED_DATA -> ver to the current TPM version.
6. Create an XOR-string by concatenating the shared session secret with the even numbered hash (generated by the TPM) and hashing the concatenated value. Generate the plaintext authorization data for the sealed data by XORing the XOR-string with the variable encAuth.
7. Set continueAuthSession to FALSE.
8. **If the data is wrapped to PCR's then**
    a. The TPM MUST check that the pcrInfo parameter is a consistent TCPA_PCR_SELECTION structure. If not, the TPM MUST return the error code TCPA_BADINDEX.
    b. The TPM MUST compute a1 by creating TCPA_COMPOSITE_HASH value using pcrInfo -> pcrSelection as the input to the algorithm in 10.4.5.
    c. The TPM MUST set TPM_STORED_DATA -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease.
    d. The TPM MUST set TPM_STORED_DATA -> SealInfo -> digestAtCreation to a1
    e. The TPM MUST set TPM_STORED_DATA -> sealInfoSize to the size of the TCPA_PCR_INFO structure.
9. Else
    a. The TPM MUST set TPM_STORED_DATA -> sealInfoSize to 0.
10. The TPM provides no validation of the authorization data. Well known values like nulls are possible and allowed.
11. The TPM must ensure that the PAYLOAD_TYPE byte of any sealed data is set to the proper value to ensure that all encrypted elements can be distinguished from each other.

### 7.2.2 TPM_Unseal

*Start of informative comment:*

The TPM_Unseal operation will reveal TPM_Sealed data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, TPM_Unseal accepts a data blob generated by a TPM_Seal operation. TPM_Unseal decrypts the structure internally, checks the integrity of the resulting data, *and* checks that the PCR named has the value named during TPM_Seal. Additionally, the caller must supply appropriate authorization data for blob and for the key that was used to seal that data.

If the integrity, platform configuration and authorization checks succeed, the sealed data is returned to the caller; otherwise, an error is generated.

*End of informative comment.*

**Type**

TPM protected capability; the user must provide authorizations to use the parent key pointed to by parentHandle.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Unseal. |
| 4 | *4* | | | TCPA_KEY_HANDLE | parentHandle | Handle of a loaded key that can unseal the data. |
| 5 | <> | 2s | <> | TCPA_STORED_DATA | inData | The encrypted data generated by TPM_Seal. |
| 6 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for parentHandle. |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | *20* | | | TCPA_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |
| 10 | *4* | | | TCPA_AUTHHANDLE | dataAuthHandle | The authorization handle used to authorize inData. |
| | | 2 H2 | *20* | TCPA_NONCE | dataLastNonceEven | Even nonce previously generated by TPM |
| 11 | *20* | 3 H2 | *20* | TCPA_NONCE | datanonceOdd | Nonce generated by system associated with entityAuthHandle |
| 12 | *1* | 4 H2 | *1* | BOOL | continueDataSession | Continue usage flag for dataAuthHandle. |
| 13 | *20* | | | TCPA_AUTHDATA | dataAuth | The authorization digest for the encrypted entity. HMAC key: entity.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Unseal. |
| 4 | 4 | 3s | 4 | UINT32 | sealedDataSize | The used size of the output area for secret |
| 5 | <> | 4s | <> | BYTE[ ] | secret | Decrypted data that had been sealed |
| 6 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |
| 9 | 20 | 2 H2 | 20 | TCPA_NONCE | dataNonceEven | Even nonce newly generated by TPM. |
| | | 3 H2 | 20 | TCPA_NONCE | datanonceOdd | Nonce generated by system associated with dataAuthHandle |
| 10 | 1 | 4 H2 | 1 | BOOL | continueDataSession | Continue use flag, TRUE if handle is still active |
| 11 | 20 | | | TCPA_AUTHDATA | dataAuth | The authorization digest used for the dataAuth session. HMAC key: entity.usageAuth. |

**Actions**

1. The TPM MUST validate that parentAuth authorizes the use of the key in parentHandle. On failure the TPM MUST return TCPA_AUTHFAIL.
2. If the keyUsage field of the key indicated by parentHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TCPA_INVALID_KEYUSAGE.
3. The TPM MUST check that the TCPA_KEY_FLAGS -> Migratable flag has the value FALSE in the key indicated by parentKeyHandle. If not, the TPM MUST return the error code TCPA_BAD_PARAMETER.
4. The TPM MUST create d1 by decrypting inData using the key pointed to by parentHandle. inData is a TCPA_STORED_DATA structure and the encrypted area is pointed to by inData -> encData.
5. The TPM MUST check the integrity of the d1. The integrity check establishes that the d1 is a consistent TPM_SEALED_DATA structure created with by a TPM_Seal operation on the same TPM that is attempting the TPM_Unseal *and* that d1 has not been modified.
    a. The TPM MUST check that the d1 -> tpmProof matches TCPA_PERSISTENT_DATA -> tpmProof.
    b. The TPM MUST calculate h1 by performing the same calculation that creates TPM_SEALED_DATA -> storedDigest.
    c. The TPM MUST validate that h1 and d1 -> storedDigest match.
    d. The TPM MUST check the TCPA_PAYLOAD_TYPE value and ensure that it is not decrypting a key.

e. If d1 fails the integrity checks, then the operation MUST return the error TCPA_NOTSEALED_BLOB.

6. The TPM must validate the authorization to use d1. The TPM MUST validate the authorization in dataAuth matches the d1 -> authData parameter. The TPM MUST return TCPA_AUTHFAIL on a mismatch.

7. **If inData is wrapped to PCR's then,**

   a. The TPM MUST ensure that the PCRs to which the blob was sealed are the same as the PCRs' values that exist at the time of TPM_Unseal.

   b. The TPM MUST validate that inData -> pcrInfo is a valid TCPA_INFO_STRUCTURE.

   c. The TPM will create h1 by computing a composite hash using the inData -> pcrInfo parameter as the input to the composite hashing algorithm (See 10.4.5).

   d. The TPM MUST compare h1 with inData -> pcrInfo -> digestAtRelease. On a mismatch the TPM MUST return TCPA_WRONGPCRVALUE.

8. **else**

   a. The TPM does not need to check PCR configuration.

### 7.2.3 TSS_Bind

*Start of informative comment*

The TSS_Bind command allows an entity outside of the TPM to create a blob that can be operated on by TPM_Unbind.

The TSS_Bind command is responsible for creating the blob to be encrypted in a manner that is decryptable by TPM_Unbind.

To bind data that is larger than the RSA public key modulus it is the responsibility of the caller to perform the blocking and subsequent combination of data.

The TSS_Bind command should perform validations that the public key presented to it is from a valid TPM.

*End of informative comment*

### 7.2.4 TPM_UnBind

*Start of informative comment:*

TPM_UnBind takes the data blob that is the result of a TSS_Bind command and decrypts it for export to the User. The caller must authorize the use of the key that will decrypt the incoming blob.

UnBind operates on a block-by-block basis, and has no notion of any relation between one block and another.

*End of informative comment.*

**Type**

TCPA protected capability; the user must provide authorization to use the key specified in the keyHandle parameter.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_UnBind. |
| 4 | *4* | | | TCPA_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can perform UnBind operations. |
| 5 | *4* | 2s | *4* | UINT32 | inDataSize | The size of the input blob |
| 6 | <> | 3s | <> | BYTE[ ] | inData | Encrypted blob to be decrypted |
| 7 | *4* | | | TCPA_AUTHHANDLE | authHandle | The handle used for keyHandle authorization |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | *20* | | | TCPA_AUTHDATA | privAuth | The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_UnBind |
| 4 | 4 | 3s | 4 | UINT32 | outDataSize | The length of the returned decrypted data |
| 5 | <> | 4s | <> | BYTE[ ] | outData | The resulting decrypted data. |
| 6 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth. |

**Description**

UnBind SHALL operate on a single block only.

**Actions**

The TPM SHALL perform the following:

1. If the inDataSize is 0 the TPM returns TCPA_BAD_PARAMETER
2. Validate the authorization to use the key pointed to by keyHandle
3. If the keyUsage field of the key referenced by keyHandle does not have the value TPM_KEY_BIND or TPM_KEY_LEGACY, the TPM must return the error code TCPA_INVALID_KEYUSAGE
4. Decrypt the inData using the key pointed to by keyHandle
5. **if (keyHandle -> encScheme does not equal TCPA_ES_RSAESOAEP_SHA1_MGF1) and (keyHandle -> keyUsage equals TPM_KEY_LEGACY),**
    a. The payload does not have TCPA specific markers to validate, so no consistency check can be performed.
    b. Set the output parameter outData to the value of the decrypted value of inData. (Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
    c. Set the output parameter outDataSize to the size of outData, as deduced from the decryption process.
    d. Return the output parameters.
6. **else**
    a. Interpret the decrypted data under the assumption that it is a TCPA_BOUND_DATA structure, and validate that the payload type is TCPA_PT_BIND
    b. Set the output parameter outData to the value of TCPA_BOUND_DATA -> payloadData. (Other parameters of TCPA_BOUND_DATA SHALL NOT be returned. Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
    c. Set the output parameter outDataSize to the size of outData, as deduced from the decryption process and the interpretation of TCPA_BOUND_DATA.

d. Return the output parameters.

### 7.2.5 TPM_CreateWrapKey

*Start of informative comment:*

The TPM_CreateWrapKey command both generates and creates a secure storage bundle for asymmetric keys.

The newly created key can be locked to a specific PCR value by specifying a set of PCR registers.

*End of informative comment.*

**Type**

TCPA protected capability; the user must provide authorization to use the key indicated by parentHandle.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateWrapKey |
| 4 | *4* | | | TCPA_KEY_HANDLE | parentHandle | Handle of a loaded key that can perform key wrapping. |
| 5 | *20* | 2s | *20* | TCPA_ENCAUTH | dataUsageAuth | Encrypted usage authorization data for the sealed data. |
| 6 | *20* | 3s | *20* | TCPA_ENCAUTH | dataMigrationAuth | Encrypted migration authorization data for the sealed data. |
| 7 | <> | 4s | <> | TCPA_KEY | keyInfo | Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. |
| 8 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for parent key authorization. Must be an OS_AP session. |
| | | 2H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | *20* | 3H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | *1* | 4H1 | *1* | BOOL | continueAuthSession | Ignored |
| 11 | *20* | | | TCPA_AUTHDATA | pubAuth | The authorization digest that authorizes the use of the public key in parentHandle. HMAC key: parentKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateWrapKey |
| 4 | <> | 4s | <> | TCPA_KEY | wrappedKey | The TCPA_KEY structure which includes the public and encrypted private key |
| 5 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed at FALSE |
| 7 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |

**Descriptions**

This command requires the encryption of two parameters. To create two XOR strings the caller combines the two nonces in use by the OSAP session with the session shared secret.

DataUsageAuth is XOR'd with the SHA-1 hash of the concatenation of the OSAP session shared secret with the even numbered nonce generated by the TPM (authLastNonceEven). MigrationAuth is XOR'd with the SHA-1 hash of the concatenation of the OSAP session shared secret with the odd numbered nonce generated by the caller (nonceOdd).

**Actions**

The TPM SHALL do the following:

1. Validate the authorization to use the key pointed to by parentHandle. Return TCPA_AUTHFAIL on any error.
2. Validate the session type for parentHandle is OS-AP.
3. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE
4. If parentHandle -> keyFlag -> migratable is TRUE and keyInfo -> keyFlag -> migratable is FALSE then return TCPA_INVALID_KEYUSAGE
5. Validate key parameters
    a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or TPM_KEY_AUTHCHANGE. If it is, return TCPA_BAD_PARAMETER.
    b. **If keyInfo -> keyUsage equals TPM_KEY_STORAGE**
        i. algorithmID MUST be TCPA_ALG_RSA
        ii. encScheme MUST be TCPA_ES_RSAESOAEP_SHA1_MGF1
        iii. sigScheme MUST be TCPA_SS_NONE
        iv. key size MUST be 2048
6. Validate all keyInfo parameters, any errors return TCPA_BAD_PARAMETER
7. Create the two XOR patterns by using the session key and the nonces for this transaction

8. Set continueAuthSession to FALSE
9. Decrypt the DataUsageAuth and DataMigrationAuth parameters
10. Generate asymmetric key according to algorithm information in keyInfo
11. Fill in the wrappedKey structure with information from the newly generated key.
    a. Set the auth member of this structure to the decrypted values of DataUsageAuth.
    b. The TPM MUST set the wrappedKey -> ver to the current TPM version.
    c. If the KeyFlags -> migratable bit is set to 1, the wrappedKey -> encData -> migrationAuth SHALL contain the decrypted value from DataMigrationAuth.
    d. If the KeyFlags -> migratable bit is set to 0, and wrappedKey -> encData -> migrationAuth SHALL be set to the value tpmProof.
12. Encrypt the private portions of the wrappedKey structure using the key in keyHandle
13. Return the newly generated key in the wrappedKey parameter

### 7.2.6 TSS_WrapKey

*Start of informative comment:*

The TSS_WrapKey command creates a migratable blob for a key that has been presented externally. The creator of the key can prevent migration by the User by wrapping it with a non-migratable storage key and loading random data for the MigrationAuthorizationData. However, the internal bit will still be set as migratable. This allows delegation of a key without giving the delegator the right to further delegate. Because the key was created elsewhere, there is no need to return the PubKey of the key being wrapped, and because a public key is used for the wrapping external to the TPM, there is no need for authorization data for the wrapping key to be passed.

*End of informative comment.*

**Actions**

The TSS SHOULD do the following:

1. If the keyUsage field of PubKey does not have the value TPM_KEY_STORAGE, the TSS must return the error code TCPA_INVALID_KEYUSAGE
2. Validate the TCPA_STORE_ASYMKEY structure
3. Fill in the TCPA_STORE_ASYMKEY structure with the authorization and usage parameters
4. Set KeyFlags.migratable to 1
5. Set all other KeyFlags members to the values in KeyFlags parameter
6. Set TCPA_STORE_ASYMKEY.pcrDigest to 20 bytes of value 0xFF.
7. Encrypt the TCPA_STORE_ASYMKEY structure using the pubkey parameter
8. Return the entire TCPA_KEY structure

### 7.2.7 TSS_WrapKeyToPcr

*Start of informative comment:*

The TSS_WrapKeyToPcr command is similar to the TSS_WrapKey command except that it has an additional requirement for authorization of use: a PCR value must match the value given at blob-creation time. Thus, TSS_WrapKeyToPcr creates a migratable blob for a key that has been presented externally. Both authorization data and a given PCR value are set as part of the authorization requirement.

*End of informative comment.*

**Actions**

The TSS SHOULD do the following:

1. If the keyUsage field of PubKey does not have the value TPM_KEY_STORAGE, the TSS must return the error code TCPA_INVALID_KEYUSAGE
2. Validate the TCPA_STORE_ASYMKEY structure
3. Fill in the TCPA_STORE_ASYMKEY structure with the authorization and usage parameters
4. Set KeyFlags.migratable to 1
5. Set all other KeyFlags members to the values in KeyFlags parameter
6. Set TCPA_STORE_ASYMKEY.pcrDigest to TargetPCRHash
7. Encrypt the TCPA_STORE_ASYMKEY structure using the pubkey parameter
8. Return the entire TCPA_KEY structure

### 7.2.8 TPM_LoadKey

*Start of informative comment:*

Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM_LoadKey function loads the key into the TPM for further use.

The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. The assumption is that the handle may change due to key management operations. It is the responsibility of upper level software to maintain the mapping between handle and any label used by external software.

The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR using parentPCRStatus.

This command has the responsibility of enforcing restrictions on the use of keys. For example, when attempting to load a STORAGE key it will be checked for the restrictions on a storage key (2048 size etc.).

The flag parentPCRStatus enables the possibility of checking that a platform passed through some particular state or states before finishing in the current state. A grandparent key could be linked to state-1, a parent key could linked to state-2, and a child key could be linked to state-3, for example. The use of the child key then indicates that the platform passed through states 1 and 2 and is currently in state 3, in this example. The issue of TPM_Startup is with stType == TCPA_ST_CLEAR is an indication that the platform has been reset, so the platform has not passed through the previous states. Hence keys with parentPCRStatus==TRUE must be unloaded if TPM_Startup is issued with stType == TCPA_ST_CLEAR.

If a TCPA_KEY structure has been decrypted AND the integrity test using "pubDataDigest" has passed AND the key is non-migratory, the key must have been created by the TPM. So there is every reason to believe that the key poses no security threat to the TPM. While there is no known attack from a rogue migratory key, there is a desire to verify that a loaded migratory key is a real key, arising from a general sense of unease about execution of arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt cycle, but this may be expensive. For RSA keys, it is therefore suggested that the consistency test consists of dividing the supposed RSA product by the supposed RSA prime, and checking that there is no remainder.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization to use the parent key pointed to by parentHandle.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_LoadKey. |
| 4 | 4 | | | TCPA_KEY_HANDLE | parentHandle | TPM handle of parent key. |
| 5 | <> | 2s | <> | TCPA_KEY | inKey | Incoming key structure, both encrypted private and clear public portions. |
| 6 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for parentHandle authorization. |
| | | 2H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 20 | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TCPA_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadKey |
| 4 | 4 | 3s | 4 | TCPA_KEY_HANDLE | inkeyHandle | Internal TPM handle where decrypted key was loaded. |
| 5 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |

### Actions

The TPM SHALL perform the following steps:

1. Validate the authorization to use the key in parentHandle
2. If the keyUsage field of the key referenced by parent handle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TCPA_INVALID_KEYUSAGE
3. Decrypt the inKey -> privkey to obtain TCPA_STORE_ASYMKEY structure using the key in parentHandle
4. Validate the integrity of inKey and decrypted TCPA_STORE_ASYMKEY
    a. Reproduce inKey -> TCPA_STORE_ASYMKEY -> pubDataDigest using the fields of inKey, and check that the reproduced value is the same as pubDataDigest
5. Validate the consistency of the key and it's key usage.
    a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the public and private components of the asymmetric key pair. If inKey -> keyFlags -> migratable is FALSE, the TPM MAY verify consistency of the public and private components of the asymmetric key pair. The consistency of an RSA key pair MAY be verified by dividing the supposed (P*Q) product by a supposed prime and checking that there is no remainder..
    b. If inKey -> keyUsage is TPM_KEY_IDENTITY, verify that inKey->keyFlags->migratable is FALSE. If it is not, return TCPA_BAD_PARAMETER
    c. If inKey -> keyUsage is TPM_KEY_AUTHCHANGE, return TCPA_BAD_PARAMETER
    d. If inKey -> keyFlags -> migratable equals 0 then verify that TCPA_STORE_ASYMKEY -> migration equals TCPA_PERSISTENT_DATA -> tpmProof
    e. Validate the mix of encryption and signature schemes according to section 4.10.1

f. **If inKey -> keyUsage is TPM_KEY_STORAGE**

i. algorithmID MUST be TCPA_ALG_RSA

ii. Key size MUST be 2048

iii. sigScheme MUST be TCPA_SS_NONE

g. **If inKey -> keyUsage is TPM_KEY_IDENTITY**

i. algorithmID MUST be TCPA_ALG_RSA

ii. Key size MUST be 2048

iii. encScheme MUST be TCPA_ES_NONE

h. **If the decrypted InKey ->pcrInfo is not NULL,**

i. The TPM validates that inKey -> pcrInfo -> pcrSelection points to at least one PCR register. If no PCR registers are selected the TPM MUST NOT perform any further checks regarding PCR registers with the loaded key.

ii. The TPM MUST store the list of active PCR registers in a manner that allows the TPM to access this list whenever the loaded key is used for any function.

iii. Every time before the loaded key is used, the inkey -> PCRInfo structure from TPM_LoadKey MUST be used to verify that the current PCR state is correct. The TPM MUST ensure that the PCRs to which the key was sealed are the same as the PCRs' values that exist at the time of key usage. To do this, the TPM will compute a TCPA_COMPOSITE_HASH value using the inkey -> pcrInfo -> pcrSelection -> pcrSelect parameter as the input to the composite hashing algorithm (See 10.4.5).

iv. If the resulting composite hash matches the inkey -> PCRInfo -> digestAtRelease parameter, the TPM is permitted to use the key. Otherwise, if the composite hashes do not match, the TPM is NOT permitted to use the key in the current PCR state, and the TPM MUST return TCPA_WRONGPCRVAL.

i. **If the decrypted inKey -> pcrInfo is NULL,**

i. The TPM MUST set the internal indicator to indicate that the key is not using any PCR registers.

6. Perform any processing necessary to make TCPA_STORE_ASYMKEY key available for operations
7. Load key and key information into internal memory of the TPM. If insufficient memory exists return error TCPA_NOSPACE.
8. Assign inKeyHandle according to internal TPM rules.
9. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
10. If ParentHandle indicates it is using PCR registers then set inKeyHandle -> parentPCRStatus to TRUE. The TPM creates an indicator of PCR usage in step 5.h.ii above. This indicator is internal to the TPM but MUST accurately reflect the sealing of a key to a PCR register.

### 7.2.9 TPM_EvictKey

**Type**

TPM command. Non-authorized.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_EvictKey |
| 4 | 4 | | | TCPA_KEY_HANDLE | evictHandle | The handle of the key to be evicted. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Actions**

The TPM will invalidate the key stored in the specified handle and return the space to the available internal pool for subsequent query by TPM_GetCapability and usage by TPM_LoadKey. If the specified key handle does not correspond to a valid key, an error will be returned.

### 7.2.10 TPM_GetPubKey

*Start of informative comment:*

The owner of a key may wish to obtain the public key value from a loaded key. This information may have privacy concerns so the command must have authorization from the key owner.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization to use the key pointed to by keyHandle.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetPubKey. |
| 4 | *4* | | | TCPA_KEY_HANDLE | keyHandle | TPM handle of key. |
| 5 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization. |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | *20* | | | TCPA_AUTHDATA | keyAuth | The authorization digest for inputs and keyHandle. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetPubKey. |
| 4 | <> | 3s | <> | TCPA_PUBKEY | pubKey | Public portion of key in keyHandle. |
| 5 | *20* | 2 H1 | *20* | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | *20* | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth. |

**Actions**

The TPM SHALL perform the following steps:

1. Validate the authorization to use the key in keyHandle
2. Create a TCPA_PUBKEY structure and return

### 7.2.11 TPM_CreateMigrationBlob

*Start of informative comment:*

The TPM_CreateMigrationBlob command implements the first step in the process of moving a migratable key to a new parent or platform. Execution of this command requires knowledge of the migrationAuth field of the key to be migrated.

Migrate mode is generally used to migrate keys from one TPM to another for backup, upgrade or to clone a key on another platform. To do this, the TPM needs to create a data blob that another TPM can deal with. This is done by loading in a backup public key that will be used by the TPM to create a new data blob for a migratable key.

The TPM Owner does the selection and authorization of migration public keys at any time prior to the execution of TPM_CreateMigrationBlob by performing the TPM_AuthorizeMigrationKey command.

ReWrap mode is used to directly move the key to a new parent (either on this platform or another). The TPM simply re-encrypts the key using a new parent, and outputs a normal encrypted element that can be subsequently used by a TPM_LoadKey command.

TPM_CreateMigrationBlob implicitly cannot be used to migrate a non-migratory key. No explicit check is required. Only the TPM knows tpmProof. Therefore it is impossible for the caller to submit an authorization value equal to tpmProof and migrate a non-migratory key.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorizations for the entity pointed to by parentHandle and inData.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateMigrationBlob |
| 4 | 4 | | | TCPA_KEY_HANDLE | parentHandle | Handle of the parent key that can decrypt encData. |
| 5 | 2 | 2s | 2 | TCPA_MIGRATE_SCHEME | migrationType | The migration type, either MIGRATE or REWRAP |
| 6 | <> | 3s | <> | TCPA_MIGRATIONKEYAUTH | migrationKeyAuth | Migration public key and its authorization digest. |
| 7 | 4 | 4s | 4 | UINT32 | encDataSize | The size of the encData parameter |
| 8 | <> | 5s | <> | BYTE[ ] | encData | The encrypted entity that is to be modified. |
| 9 | 4 | | | TCPA_AUTHHANDLE | parentAuthHandle | The authorization handle used for the parent key. |
| | | 2H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 10 | 20 | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 11 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag for parent session |
| 12 | 20 | | 20 | TCPA_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |

| | | | | | | |
|---|---|---|---|---|---|---|
| 13 | 4 | | | TCPA_AUTHHANDLE | entityAuthHandle | The authorization handle used for the encrypted entity. |
| | | 2H2 | 20 | TCPA_NONCE | entitylastNonceEven | Even nonce previously generated by TPM |
| 14 | 20 | 3 H2 | 20 | TCPA_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 15 | 1 | 4 H2 | 1 | BOOL | continueEntitySession | Continue use flag for entity session |
| 16 | 20 | | | TCPA_AUTHDATA | entityAuth | The authorization digest for the inputs and encrypted entity. HMAC key: entity.migrationAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateMigrationBlob |
| 4 | 4 | 3s | 4 | UINT32 | randomSize | The used size of the output area for random |
| 5 | <> | 4s | <> | BYTE[ ] | random | String used for xor encryption |
| 6 | 4 | 5s | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 7 | <> | 6s | <> | BYTE[ ] | outData | The modified, encrypted entity. |
| 8 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 4 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 9 | 1 | 5 H1 | 1 | BOOL | continueAuthSession | Continue use flag for parent key session |
| 10 | 20 | | 20 | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters and parentHandle. HMAC key: parentKey.usageAuth. |
| 11 | 20 | 3 H2 | 20 | TCPA_NONCE | entityNonceEven | Even nonce newly generated by TPM to cover entity |
| | | 4 H2 | 20 | TCPA_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 12 | 1 | 5 H2 | 1 | BOOL | entityContinueAuthSession | Continue use flag for entity session |
| 13 | 20 | | | TCPA_AUTHDATA | entityAuth | The authorization digest for the returned parameters and entity. HMAC key: entity.migrationAuth. |

**Description**

The key that wraps the migration key MUST be a 2048 bit RSA key or higher.

The TPM does not check the PCR values when migrating values locked to a PCR.

The second authorisation session (using entityAuth) MUST be OIAP because OSAP does not have a suitable entityType

**Actions**

1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
2. Create d1 by decrypting encData using the key pointed to by parentHandle.
3. Validate that entityAuth authorizes the migration of d1. The validation MUST use d1 -> migrationAuth as the secret.
4. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
5. **If migrationType == TCPA_MS_MIGRATE the TPM SHALL perform the following actions:**
    a. Build a TCPA_STORE_PRIVKEY structure from the d1 key. This privKey element should be 132 bytes long for a 2K RSA key.
    b. Create k1 and k2 by splitting the privKey element created in step a into 2 parts. k1 is the first 20 bytes of privKey, k2 contains the remainder of privKey.
    c. Build m by filling in the usageAuth and pubDataDigest fields within a TCPA_MIGRATE_ASYMKEY structure using data from the d1 key. The privKey field should be set to k2 (step g) and payload should be set to TCPA_PT_MIGRATE.
    d. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters of
        i. *m* = TCPA_MIGRATE_ASYMKEY structure (step c)
        ii. *pHash* = d1->migrationAuth
        iii. *seed* = *s1* = k1 (step g)
    e. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1. Return r1 in the Random parameter.
    f. Create x1 by XOR of o1 with r1
    g. Copy r1 into the output field "random".
    h. Encrypt x1 with the migration public key included in migrationKeyAuth.
6. **If migrationType == TCPA_MS_REWRAP the TPM SHALL perform the following actions:**
    a. Rewrap the key using the public key in migrationKeyAuth, keeping the existing contents of that key.
    b. If randomSize is 0 the TPM returns TCPA_BAD_PARAMETER.

### 7.2.12 TPM_ConvertMigrationBlob

*Start of informative comment:*

This command takes a migration blob and creates a normal wrapped blob. The migrated blob must be loaded into the TPM using the normal TPM_LoadKey function.

Note that the command migrates private keys, only. The migration of the associated public keys is not specified by TCPA because they are not security sensitive. Migration of the associated public keys may be specified in a platform specific specification. A TCPA_KEY structure must be recreated before the migrated key can be used by the target TPM in a LoadKey command.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization to use the key in parentHandle

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ConvertMigrationBlob. |
| 4 | 4 | | | TCPA_KEY_HANDLE | parentHandle | Handle of a loaded key that can decrypt keys. |
| 5 | 4 | 2s | 4 | UINT32 | inDataSize | Size of inData |
| 6 | <> | 3s | <> | BYTE [ ] | inData | The XOR'd and encrypted key |
| 7 | 4 | 4s | 4 | UINT32 | randomSize | Size of random |
| 8 | <> | 5s | <> | BYTE [ ] | random | Random value used to hide key data. |
| 9 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 10 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 11 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 12 | 20 | | | TCPA_AUTHDATA | parentAuth | The authorization digest that authorizes the inputs and the migration of the key in parentHandle. HMAC key: parentKey.usageAuth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ConvertMigrationBlob |
| 4 | 4 | 3s | 4 | UINT32 | outDataSize | The used size of the output area for outData |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | <> | 4S | <> | BYTE[ ] | outData | The encrypted private key that can be loaded with TPM_LoadKey |
| 6 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth |

**Action**

The TPM SHALL perform the following:

1. Validate the authorization to use the key in parentHandle
2. If the keyUsage field of the key referenced by parentHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TCPA_INVALID_KEYUSAGE
3. Create d1 by decrypting the inData area using the key in parentHandle
4. Create o1 by XOR d1 and random parameter
5. Create m1, seed and pHash by OAEP decoding o1
6. Verify that the payload type is TCPA_PT_MIGRATE
7. Create k1 by combining seed and the TCPA_MIGRATE_ASYMKEY.data field
8. Create d2 a TCPA_STORE_ASYMKEY structure by inserting pHash as the migration authorization field. Set the TCPA_STORE_ASYMKEY -> privKey field to k1
9. Create outData using the key in parentHandle to perform the encryption

### 7.2.13 TPM_AuthorizeMigrationKey

*Start of informative comment:*

This command creates an authorization blob, to allow the TPM owner to specify which migration facility they will use and allow users to migrate information without further involvement with the TPM owner.

The TPM does no validation of the migration key. It is the responsibility of the TPM Owner to determine the validity of the key and whether it is appropriate for use by the TPM.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization from the TPM Owner

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_AuthorizeMigrationKey |
| 4 | *2* | 2s | *2* | TCPA_MIGRATE_SCHEME | migrateScheme | Type of migration operation that is to be permitted for this key. |
| 4 | <> | 3s | <> | TCPA_PUBKEY | migrationKey | The public key to be authorized. |
| 5 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | *20* | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_AuthorizeMigrationKey |
| 4 | <> | 3s | <> | TCPA_MIGRATIONKEYAUTH | outData | Returned public key and authorization digest. |
| 5 | *20* | 2 H1 | *20* | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | *20* | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Action**

The TPM SHALL perform the following:

1. Validate the authorization to use the TPM by the TPM Owner
2. Create a f1 a TCPA_MIGRATIONKEYAUTH structure
3. Set f1 -> migrationKey to the input migrationKey
4. Set f1 -> migrationScheme to the input migrationScheme
5. Create v1 by concatenating (migrationKey || migrationScheme || TCPA_PERSISTENT_DATA -> tpmProof)
6. Create h1 by performing a SHA1 hash of v1
7. Set f1 -> digest to h1
8. Return f1 as outData

## 7.3 TPM Optional Functions: Maintenance

*Start of informative comment:*

Maintenance is different from backup/migration, because maintenance provides for the migration of both migratory and non-migratory data. Maintenance is an optional TPM function, but if a TPM enables maintenance, the maintenance capabilities in this specification are mandatory – no other migration capabilities shall be used. Maintenance necessarily involves the manufacturer of a Subsystem.

When maintaining computer systems, it is sometimes the case that a manufacturer or its representative needs to replace a Subsystem containing a TPM. Some manufacturers consider it a requirement that there be a means of doing this replacement without the loss of the non-migratable keys held by the original TPM.

The owner and users of TCPA platforms need assurance that the data within protected storage is adequately protected against interception by third parties or the manufacturer.

This process MUST only be performed between two platforms of the same manufacturer and model. If the maintenance feature is supported, this section defines the required functions defined at a high level. The final function definitions and entire maintenance process is left to the manufacturer to define within the constraints of these high-level functions.

Any maintenance process must have certain properties. Specifically, any migration to a replacement Subsystem must require collaboration between the Owner of the existing Subsystem and the manufacturer of the existing Subsystem. Further, the procedure must have adequate safeguards to prevent a non-migratable key being transferred to multiple Subsystems.

The maintenance capabilities TPM_CreateMaintenanceArchive and TPM_LoadMaintenanceArchive enable the transfer of all Protected Storage data from a Subsystem containing a first TPM ($TPM_1$) to a Subsystem containing a second TPM ($TPM_2$).

A manufacturer places a public key in non-volatile storage into its TPMs at manufacture time.

The Owner of $TPM_1$ uses TPM_CreateMaintenanceArchive to create a maintenance archive that enables the migration of all data held in Protected Storage by $TPM_1$. The Owner of $TPM_1$ must provide his or her authorization to the Subsystem. The TPM then creates the TCPA_MIGRATE_ASYMKEY structure and follows the process defined.

The XOR process prevents the manufacturer from ever obtaining plaintext $TPM_1$ data.

The additional random data provides a means to assure that a maintenance process cannot subvert archive data and hide such subversion.

The random mask can be generated by two methods, either using the TPM RNG or MGF1 on the TPM Owners authorization data.

The manufacturer takes the maintenance blob, decrypts it with its private key, and satisfies itself that the data bundle represents data from that Subsystem manufactured by that manufacturer. Then the manufacturer checks the endorsement certificate of $TPM_2$ and verifies that it represents a platform to which data from $TPM_1$ may be moved.

The manufacturer dispatches two messages.

The first message is made available to CAs, and is a revocation of the $TPM_1$ endorsement certificate.

The second message is sent to the Owner of $TPM_2$, which will communicate the SRK, tpmProof and the manufacturers permission to install the maintenance blob only on $TPM_2$.

The Owner uses TPM_LoadMaintenanceArchive to install the archive copy into $TPM_2$, and overwrite the existing $TPM_2$-SRK and $TPM_2$-tpmProof in $TPM_2$. $TPM_2$ overwrites $TPM_2$-SRK with $TPM_1$-SRK, and overwrites $TPM_2$-tpmProof with $TPM_1$-tpmProof.

Note that the command TPM_KillMaintenanceFeature prevents the operation of TPM_CreateMaintenanceArchive and TPM_LoadMaintenanceArchive. This enables an Owner to block maintenance (and hence the migration of non-migratory data) either to or from a TPM.

It is required that a manufacturer takes steps that prevent further access of migrated data by TPM. This may be achieved by deleting the existing Owner from TPM, for example.

For the manufacturer to validate that the maintenance blob is coming from a valid TPM the manufacturer can require that a TPM identity sign the maintenance blob. The identity would be from a CA under the control of the manufacturer and hence the manufacturer would be satisfied that the blob is from a valid TPM.

*End of informative comment.*

Any migration of non-migratory data protected by a Subsystem SHALL require the cooperation of both the Owner of that non-migratory data and the manufacturer of that Subsystem. That manufacturer SHALL NOT cooperate in a maintenance process unless the manufacturer is satisfied that non-migratory data will exist in exactly one Subsystem. A TPM SHALL NOT provide capabilities that support migration of non-migratory data unless those capabilities are described in the TCPA specification.

The maintenance feature MUST move the following

- TCPA_KEY for SRK. The maintenance process will reset the SRK authorization to match the TPM Owners authorization
- TCPA_PERSISTENT_DATA -> tpmProof
- TPM Owners authorization

### 7.3.1 TPM_CreateMaintenanceArchive

*Start of informative comment:*

This command creates the MaintenanceArchive. It can only be executed by the owner, and may be shut off with the TPM_KillMaintenanceFeature command.

*End of informative comment.*

**Type**

Optional; TCPA protected capability; user must provide authentication from the TPM Owner.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Cmd ordinal: TPM_ORD_CreateMaintenanceArchive |
| 4 | *1* | 2s | *1* | BOOL | generateRandom | Use RNG or Owner auth to generate 'random'. |
| 5 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | *20* | 3H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | *1* | 4H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | *20* | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | *4* | TCPA_COMMAND_CODE | ordinal | Cmd ordinal: TPM_ORD_CreateMaintenanceArchive |
| 4 | *4* | 3s | *4* | UINT32 | randomSize | Size of the returned random data. Will be 0 if generateRandom is FALSE. |
| 5 | <> | 4s | <> | BYTE [ ] | random | Random data to XOR with result. |
| 6 | *4* | 5s | *4* | UINT32 | archiveSize | Size of the encrypted archive |
| 7 | <> | 6s | <> | BYTE [ ] | archive | Encrypted key archive. |
| 8 | *20* | 2H1 | *20* | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | *1* | 4H1 | *1* | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 10 | *20* | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

### Actions

Upon authorization being confirmed this command does the following:

1. Validates that the TCPA_PERSISTENT_FLAGS -> AllowMaintenance is TRUE.
2. Validates the TPM Owner authorization.
3. If the value of TCPA_PERSISTENT_DATA -> ManuMaintPub is zero, the TPM MUST return the error code TCPA_KEYNOTFOUND
4. Build a1 a TCPA_KEY structure using the SRK. The encData field is not a normal TCPA_STORE_ASYMKEY structure but rather a TCPA_MIGRATE_ASYMKEY structure built using the following actions.
5. Build a TCPA_STORE_PRIVKEY structure from the SRK. This privKey element should be 132 bytes long for a 2K RSA key.
6. Create k1 and k2 by splitting the privKey element created in step 4 into 2 parts. k1 is the first 20 bytes of privKey, k2 contains the remainder of privKey.
7. Build m1 by creating and filling in a TCPA_MIGRATE_ASYMKEY structure
    a. m1 -> usageAuth is set to TCPA_PERSISTENT_FIELDS -> tmpProof
    b. m1 -> pubDataDigest is set to the digest value of the SRK fields from step 4
    c. m1 -> payload is set to TCPA_PT_MAINT
    d. m1 -> partPrivKey is set to k2
8. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters of
    a. *m* = TCPA_MIGRATE_ASYMKEY structure (step 7)
    b. *P* = TCPA_PERSISTENT_FIELDS -> ownerAuth
    c. *seed* = *s1* = k1 (step 6)
9. If GenerateRandom = TRUE
    a. Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same size as o1. Set RandomData parameter to r1
10. If GenerateRandom = FALSE
    a. Create r1 by applying MGF1 to the TPM Owner authorization data. The size of r1 MUST be the same size as o1. Set RandomData parameter to null.
11. Create x1 by XOR of o1 with r1
12. Encrypt x1 with the ManuMaintPub key using the TCPA_ES_RSAESOAEP_SHA1_MGF1 encryption scheme.
13. Set a1 -> encData to x1
14. Return a1 in the archive parameter

### 7.3.2 TPM_LoadMaintenanceArchive

*Start of informative comment:*

This command loads in a Maintenance archive that has been massaged by the manufacturer to load into another TPM.

*End of informative comment.*

#### Type

Optional; TCPA protected capability; user must provide authentication from the TPM Owner.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1 | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadMaintenanceArchive |
| | | | | … | … | Vendor specific arguments |
| - | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | - | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| - | 20 | - | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| - | 1 | - | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| – | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1 | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2 | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadMaintenanceArchive |
| | | | | .. | .. | Vendor specific arguments |
| - | 20 | - | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | - | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| - | 1 | - | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| - | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

#### Descriptions

The maintenance mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of maintenance. The TPM MUST NOT use the endorsement key for identification or encryption in the maintenance process. The maintenance process MAY use a TPM Identity to deliver maintenance information to specific TPM's.

The maintenance process can only change the SRK, tpmProof and TPM Owner authorization fields.

The maintenance process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TCPA specification, protection profiles and security targets after maintenance. The maintenance MAY NOT decrease the security values from the original security target.

The security target used to evaluate this TPM MUST include this command in the TOE.

**Actions**

The TPM SHALL perform the following when executing the command

1. Validate the TPM Owner's authorization
2. Validate that the maintenance information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
3. The packet MUST contain m2 as defined in 7.3.1
4. Ensure that only the target TPM can interpret the maintenance packet. The protection mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
5. Process the maintenance information and update the SRK and TCPA_PERSISTENT_DATA -> tpmProof fields.
6. Set the SRK useageAuth to be the same as TPM Owners authorization

### 7.3.3 TPM_KillMaintenanceFeature

*Informative Comments:*

The KillMaintenanceFeature is a permanent action that prevents ANYONE from creating a maintenance archive. This action, once taken, is permanent until a new TPM Owner is set.

This action is to allow those customers who do not want the maintenance feature to not allow the use of the maintenance feature.

At the discretion of the Owner, it should be possible to kill the maintenance feature in such a way that the only way to recover maintainability of the platform would be to wipe out the root keys. This feature is mandatory in any TPM that implements the maintenance feature.

*End informative Comment*

**Type**

Optional; TCPA protected capability; user must provide authentication from the TPM Owner.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_KillMaintenanceFeature |
| 4 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_KillMaintenanceFeature |
| 4 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. Validate the TPM Owner authorization
2. Set the TCPA_PERSISTENT_FLAGS.AllowMaintenance flag to FALSE.

### 7.3.4 TPM_LoadManuMaintPub

*Informative Comments:*

The LoadManuMaintPub command loads the manufacturer's public key for use in the maintenance process. The command installs ManuMaintPub in persistent data storage inside a TPM. Maintenance enables duplication of non-migratory data in protected storage. There is therefore a security hole if a platform is shipped before the maintenance public key has been installed in a TPM.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

*End of Informative Comments*

Incoming Operands and Sizes

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadManuMaintPub |
| 4 | 20 | | | TCPA_NONCE | antiReplay | AntiReplay and validation nonce |
| 5 | <> | | | TCPA_PUBKEY | pubKey | The public key of the manufacturer to be in use for maintenance |

Outgoing Operands and Sizes

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadManuMaintPub |
| 4 | 20 | | | TCPA_DIGEST | checksum | Digest of pubKey and antiReplay |

**Type**

Optional; TCPA protected capability

**Description**

The pubKey MUST specify an algorithm whose strength is not less than the RSA algorithm with 2048bit keys.

pubKey SHOULD unambiguously identify the entity that will perform the maintenance process with the TPM Owner.

TCPA_PERSISTENT_DATA -> ManuMaintPub SHALL exist in a TCPA-shielded location, only.

If an entity (Platform Entity) does not support the maintenance process but issues a platform credential for a platform containing a TPM that supports the maintenance process, the value of TCPA_PERSISTENT_DATA -> ManuMaintPub MUST be set to zero before the platform leaves the entity's control.

### Actions

The first valid TPM_LoadManuMaintPub command received by a TPM SHALL

1. Store the parameter pubKey as TCPA_PERSISTENT_DATA -> ManuMaintPub.
2. Create "checksum" by concatenating data to form (pubKey||antiReplay) and passing the concatenated data through a SHA-1 hash process.
3. Export the checksum

Subsequent calls to TPM_LoadManuMaintPub SHALL return code TCPA_FAIL.

### 7.3.5 TPM_ReadManuMaintPub

***Informative Comments:***

The ReadManuMaintPub command is used to check whether the manufacturer's public maintenance key in a TPM has the expected value. This may be useful during the manufacture process. The command returns a digest of the installed key, rather than the key itself. This hinders discovery of the maintenance key, which may (or may not) be useful for manufacturer privacy.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

***End of Informative Comments***

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReadManuMaintPub |
| 4 | *20* | | | TCPA_NONCE | antiReplay | AntiReplay and validation nonce |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReadManuMaintPub |
| 4 | *20* | | | TCPA_DIGEST | checksum | Digest of pubKey and antiReplay |

**Type**

Optional; TCPA protected capability

**Description**

This command returns the hash of the antiReplay nonce and the previously loaded manufacturer's maintenance public key.

**Actions**

The TPM_ ReadManuMaintKey command SHALL

1. Create "checksum" by concatenating data to form (TCPA_PERSISTENT_DATA -> ManuMaintPub ||antiReplay) and passing the concatenated data through SHA1.
2. Export the checksum

# 8. Cryptographic and Miscellaneous Functions

## 8.1 Introduction

*Start of informative comment:*

This section describes the cryptographic functions and the miscellaneous functions that do not fit into any specific category.

*End of informative comment.*

## 8.2 TPM Hash Operations

*Start of informative comment:*

The TPM must provide support to produce a SHA-1 digest. These commands are primarily intended for use in the early stages of a boot process, before more sophisticated computing resources are available.

*End of informative comment.*

The only commands that SHALL be presented to the TPM in-between a TPM_SHA1Start command and a TPM_SHA1Complete command SHALL be a variable number (possibly 0) of TPM_SHA1Update commands.

The only commands that SHALL be presented to the TPM in-between a TPM_SHA1Start command and a TPM_SHA1CompleteExtend command SHALL be a variable number (possibly 0) of TPM_SHA1Update commands.

### 8.2.1 TPM_SHA1Start

*Start of informative comment:*

This capability starts the process of calculating a SHA-1 digest.

*End of informative comment.*

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1Start |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | UINT32 | maxNumBytes | Maximum number of bytes that can be sent to TPM_SHA1Update. Must be a multiple of 64 bytes. |

**Description**

This capability prepares the TPM for a subsequent TPM_SHA1Update, TPM_SHA1Complete or TPM_SHA1CompleteExtend command. The capability SHALL open a thread that calculates a SHA-1 digest.

### 8.2.2 TPM_SHA1Update

*Start of informative comment:*

This capability inputs complete blocks of data into a pending SHA-1 digest. At the end of the process, the digest remains pending.

*End of informative comment.*

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1Update |
| 4 | 4 | | | UINT32 | numBytes | The number of bytes in hashData. Must be a multiple of 64 bytes. |
| 5 | <> | | | BYTE [ ] | hashData | Bytes to be hashed |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Description**

This command SHALL incorporate complete blocks of data into the digest of an existing SHA-1 thread. Only integral numbers of complete blocks (64 bytes each) can be processed.

### 8.2.3 TPM_SHA1Complete

Start of informative comment:

This capability terminates a pending SHA-1 calculation.

End of informative comment.

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1Complete |
| 4 | *4* | | | UINT32 | hashDataSize | Number of bytes in hashData, MUST be 64 or less |
| 5 | <> | | | BYTE [ ] | hashData | Final bytes to be hashed |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | *20* | | | TCPA_DIGEST | hashValue | The output of the SHA-1 hash. |

**Description**

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, and terminate that thread. hashDataSize MAY have values in the range of 0 through 64, inclusive.

### 8.2.4 TPM_SHA1CompleteExtend

Start of informative comment:

This capability terminates a pending SHA-1 calculation and EXTENDS the result into a Platform Configuration Register using a SHA-1 hash process.

This command is designed to complete a hash sequence and extend a PCR in memory-less environments.

End of informative comment.

#### Type

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1CompleteExtend |
| 4 | 4 | | | TCPA_PCRINDEX | pcrNum | Index of the PCR to be modified |
| 5 | 4 | | | UINT32 | hashDataSize | Number of bytes in hashData, MUST be 64 or less |
| 6 | <> | | | BYTE [ ] | hashData | Final bytes to be hashed |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 20 | | | TCPA_DIGEST | hashValue | The output of the SHA-1 hash. |
| 5 | 20 | | | TCPA_PCRVALUE | outDigest | The PCR value after execution of the command. |

#### Description

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, EXTEND the resultant digest into a PCR, and terminate the thread. hashDataSize MAY have values in the range of 0 through 64, inclusive.

## 8.3 Key Certification

### 8.3.1 TPM_CertifyKey

*Start of informative comment:*

The TPM_CERTIFYKEY operation allows a key to certify the public portion of certain storage and signing keys.

A TPM identity key may be used to certify non-migratable keys but is not permitted to certify migratory keys. As such, it allows the TPM to make the statement "this key is held in a TCPA-shielded location, and it will never be revealed." For this statement to have veracity, the Challenger must trust the policies used by the Privacy CA that issued the identity and the maintenance policy of the TPM manufacturer.

Signing and legacy keys may be used to certify both migratable and non-migratable keys. Then the usefulness of a certificate depends on the trust in the certifying key by the recipient of the certificate.

The key to be certified must be loaded before TPM_CertifyKey is called.

See appendix B for a table of where and when keys are in use.

*End of informative comment.*

**Type**

TCPA protected capability; user must authorize the use of key pointed to by idHandle and the key pointed to by keyHandle.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_CertifyKey |
| 4 | *4* | | | TCPA_KEY_HANDLE | certHandle | Handle of the key to be used to certify the key. |
| 5 | *4* | | | TCPA_KEY_HANDLE | keyHandle | Handle of the key to be certified. |
| 6 | 20 | 2s | *20* | TCPA_NONCE | antiReplay | 160 bits of externally supplied data (typically a nonce provided to prevent replay-attacks) |
| 7 | *4* | | | TCPA_AUTHHANDLE | certAuthHandle | The authorization handle used for certHandle. |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with certAuthHandle |
| 9 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | *20* | | | TCPA_AUTHDATA | certAuth | The authorization digest for inputs and certHandle. HMAC key: certKey.auth. |
| 11 | *4* | | | TCPA_AUTHHANDLE | keyAuthHandle | The authorization handle used for the key to be signed. |
| | | 2 H2 | *20* | TCPA_NONCE | keylastNonceEven | Even nonce previously generated by TPM |
| 12 | *20* | 3 H2 | *20* | TCPA_NONCE | keynonceOdd | Nonce generated by system associated with keyAuthHandle |

| | | | | | | |
|---|---|---|---|---|---|---|
| 13 | 1 | 4 H2 | 1 | BOOL | continueKeySession | The continue use flag for the authorization handle |
| 14 | 20 | | | TCPA_AUTHDATA | keyAuth | The authorization digest for the inputs and key to be signed. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| *Param* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *Sz* | *#* | *Sz* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal TPM_ORD_CertifyKey |
| 4 | 95 | 3s | 95 | TCPA_CERTIFY_INFO | certifyInfo | The certifyInfo structure that corresponds to the signed key. |
| 5 | 4 | 4s | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 6 | <> | 5s | <> | BYTE[ ] | outData | The signed public key. |
| 7 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with certAuthHandle |
| 8 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag for cert key session |
| 9 | 20 | | 20 | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters and parentHandle. HMAC key: certKey -> auth. |
| 10 | 20 | 2 H2 | 20 | TCPA_NONCE | keyNonceEven | Even nonce newly generated by TPM |
| | | 3 H2 | 20 | TCPA_NONCE | keynonceOdd | Nonce generated by system associated with keyAuthHandle |
| 11 | 1 | 4 H2 | 1 | BOOL | continueKeyAuthSession | Continue use flag for target key session |
| 12 | 20 | | | TCPA_AUTHDATA | keyAuth | The authorization digest for the target key. HMAC key: key.auth. |

## Actions

1. The TPM validates that the key pointed to by certHandle has a signature scheme of TCPA_SS_RSASSAPKCS1v15_SHA1.
2. The TPM verifies the authorization in certAuthHandle provides authorization to use the key pointed to by certHandle.
3. The TPM verifies the authorization in keyAuthHandle provides authorization to use the key pointed to by keyHandle.
4. If the key pointed to by certHandle is an identity key (certHandle:TCPA_KEY -> keyUsage is TPM_KEY_IDENTITY), the TPM verifies that the key pointed to by keyHandle is a non-migratory key.
5. The TPM SHALL create a c1 a TCPA_CERTIFY_INFO (defined in section 4.28) structure from the key pointed to by keyHandle.
6. The TPM calculates the digest of the (public key) keyHandle -> pubKey -> key and stores it in the c1 -> pubkeyDigest.

7. The TPM copies the antiReplay parameter to the TCPA_CERTIFY_INFO c1 -> data.
8. **If pcrInfoSize is not 0 for the key pointed by keyHandle,**
    a. The TPM MUST set c1 -> pcrInfoSize to match the pcrInfoSize from the keyHandle key.
    b. The TPM MUST set c1 -> pcrInfo to match the pcrInfo from the keyHandle key.
    c. The TPM MUST set c1 -> digestAtCreation to 20 bytes of 0x00.
9. **If pcrInfoSize is 0 for the key pointed to by keyHandle**
    a. The TPM MUST set c1 -> pcrInfoSize to 0
10. The TPM creates m1, a message digest formed by taking the SHA1 of c1.
11. The TPM then performs a signature using certHandle -> sigScheme. The resulting signed blob is returned in outData.

## 8.4 TPM Internal Asymmetric Encryption

*Start of Informative Comment:*

For asymmetric encryption schemes the TPM is not required to perform the blocking of information where that information cannot be encrypted in a single cryptographic operation. The schemes TCPA_ES_RSAESOAEP_SHA1_MGF1 and TCPA_ES_RSAESPKCSv15 allow only single block encryption. When using these schemes the caller to the TPM must perform any blocking and unblocking outside the TPM. It is the responsibility of the caller to ensure that multiple blocks are properly protected using a chaining mechanism.

Note that there are inherent dangers associated with splitting information so that it can be encrypted in multiple blocks with an asymmetric key and then chaining together these blocks together. For example if an integrity check mechanism is not used, an attacker can encrypt his own data using the public key and substitute this rogue block for one of the original blocks in the message, thus forcing the TPM to replace part of the message upon decryption.

There is also a more subtle attack to discover the data encrypted in low-entropy blocks. The attacker makes a guess at the plaintext data, encrypts it, and substitutes the encrypted guess for the original block. When the TPM decrypts the complete message, a successful decryption will indicate that his guess was correct.

There are a number of solutions which could be considered for this problem – One such solution for TPMs supporting symmetric encryption is specified in PKCS#7, section 10, and involves using the public key to encrypt a symmetric key, then using that symmetric key to encrypt the long message.

For TPMs without symmetric encryption capabilities, an alternative solution may be to add random padding to each message block, thus increasing the block's entropy.

*End of informative comment*

The TPM MUST check that the encryption scheme defined for use with the key is a valid scheme for the key type, as follows:

| Key algorithm | Approved schemes | Scheme Value |
|---|---|---|
| TCPA_ALG_RSA | TCPA_ES_NONE | 0x0001 |
| | TCPA_ES_RSAESPKCSv15 | 0x0002 |
| | TCPA_ES_RSAESOAEP_SHA1_MGF1 | 0x0003 |

For a TPM_UNBIND command where the parent key has pubKey.algorithmId equal to TCPA_ALG_RSA and pubKey.encScheme set to TCPA_ES_RSAESPKCSv15 the TPM SHALL NOT expect a PAYLOAD_TYPE structure to pre-pend the decrypted data.

The TPM MUST perform the encryption or decryption in accordance with the specification of the encryption scheme, as described below.

When a null terminated string is included in a calculation, the terminating null SHALL NOT be included in the calculation.

### 8.4.1 TCPA_ES_RSAESOAEP_SHA1_MGF1

The encryption and decryption MUST be performed using the scheme RSA_ES_OAEP defined in [PKCS #1v2.0: 8.1] using SHA1 as the hash algorithm for the encoding operation.

1. **Encryption**
   a. The OAEP encoding P parameter MUST be the NULL terminated string "TCPA".
   b. If there is an error with the encryption the TPM must return the error TCPA_ENCRYPT_ERROR.
2. **Decryption**
   a. The OAEP decoding P parameter MUST be the NULL terminated string "TCPA".
   b. If there is an error with the decryption, the TPM must return the error TCPA_DECRYPT_ERROR.

### 8.4.2 TCPA_ES_RSAESPKCSV15

The encryption MUST be performed using the scheme RSA_ES_PKCSV15 defined in [PKCS #1v2.0: 8.1].

1. **Encryption**
   a. If there is an error with the encryption, return the error TCPA_ENCRYPT_ERROR.
2. **Decryption**
   a. If there is an error with the decryption, return the error TCPA_DECRYPT_ERROR.

## 8.5 TPM Internal Digital Signatures

*Start of informative comment:*

These values indicate the approved schemes in use by the TPM to generate digital signatures.

*End of informative comment.*

The TPM MUST check that the signature scheme defined for use with the key is a valid scheme for the key type, as follows:

| Key algorithm | Approved schemes | Scheme Value |
|---|---|---|
| TCPA_ALG_RSA | TCPA_SS_NONE | 0x0001 |
| | TCPA_SS_RSASSAPKCS1v15_SHA1 | 0x0002 |
| | TCPA_SS_RSASSAPKCS1v15_DER | 0x0003 |

The TPM MUST perform the signature or verification in accordance with the specification of the signature scheme, as described below.

### 8.5.1 TCPA_SS_RSASSAPKCS1v15_SHA1

The signature MUST be performed using the scheme RSASSA-PKCS1-v1.5 defined in [PKCS #1v2.0: 8.1] using SHA1 as the hash algorithm for the encoding operation.

### 8.5.2 TCPA_SS_RSASSAPKCS1v15_DER

The signature MUST be performed using the scheme RSASSA-PKCS1-v1.5 defined in [PKCS #1v2.0: 8.1]. The caller must properly format the area to sign using the DER rules. The provided area maximum size is $k$-11 octets.

## 8.6 HMAC Calculation

*Start of informative comment:*

The HMAC provides two pieces of information to the TPM: proof of knowledge of the authorization data and proof that the request arriving is authorized and has no modifications made to the command in transit.

The HMAC definition is for the HMAC calculation only. It does not specify the order or mechanism that transports the data from caller to actual TPM.

The creation of the HMAC is order dependent. Each command has specific items that are portions of the HMAC calculation. The actual calculation starts with the definition from RFC 2104.

RFC 2104 requires the selection of two parameters to properly define the HMAC in use. These values are the key length and the block size. This specification will use a key length of 20 bytes and a block size of 64 bytes. These values are known in the RFC as K for the key length and B as the block size.

The basic construct is:

H(K XOR opad, H(K XOR ipad, text))

where

- H = the SHA1 hash operation
- K = the key or the authorization data
- XOR = the XOR operation
- opad = the byte 0x5C repeated B times
- B = the block length
- ipad = the byte 0x36 repeated B times
- text = the message information and any parameters from the command

*End of informative comment.*

The TPM MUST support the calculation of an HMAC according to RFC 2104.

The size of the key (K in RFC 2104) MUST be 20 bytes. The block size (B in RFC 2104) MUST be 64 bytes.

The order of the parameters is critical to the TPM's ability to recreate the HMAC. Not all of the fields are sent on the wire for each command for instance only one of the nonce values travels on the wire. The order of the parameters is set by section 4.4.

Each function indicates what parameters are involved in the HMAC calculation.

## 8.7 Digital Signatures

### 8.7.1 TPM_Sign

*Start of informative comment:*

The Sign command signs data and returns the resulting digital signature.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization to use the keyHandle parameter.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Sign. |
| 4 | 4 | | | TCPA_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can perform digital signatures. |
| 5 | 4 | 2s | 4 | UINT32 | areaToSignSize | The size of the areaToSign parameter |
| 6 | <> | 3s | <> | BYTE[] | areaToSign | The value to sign |
| 7 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TCPA_AUTHDATA | privAuth | The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Sign. |
| 4 | 4 | 3s | 4 | UINT32 | sigSize | The length of the returned digital signature |
| 5 | <> | 4s | <> | BYTE[ ] | sig | The resulting digital signature. |
| 6 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Description**

The TPM MUST support all values of areaToSignSize that are legal for the defined signature scheme and key size. The maximum value of areaToSignSize is determined by the defined signature scheme and key size. In the case of PKCS1v15_SHA1 the areaToSignSize MUST be TCPA_DIGEST (the hash size of a sha1 operation - see 8.5.1 TCPA_SS_RSASSAPKCS1v15_SHA1). In the case of PKCS1v15_DER the maximum size of areaToSign is k-11 octets, where k is limited by the key size (see 8.5.2 TCPA_SS_RSASSAPKCS1v15_DER).

**Actions**

1. If the areaToSignSize is 0 the TPM returns TCPA_BAD_PARAMETER.
2. The TPM validates the authorization to use the key pointed to by keyHandle.
3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGN or TPM_KEY_LEGACY, if not return the error code TCPA_INVALID_KEYUSAGE
4. The TPM verifies that the signature scheme used by the key referenced by keyHandle is a valid and supported signature scheme.
5. The TPM verifies that the signature scheme and key size can properly sign the areaToSign parameter.
6. The TPM computes the signature, sig, using the key referenced by keyHandle, using with areaToSign as the information to be signed.

### 8.7.2 TSS_VerifySignature

*Start of informative comment:*

VerifySignature takes a hash and verifies the digital signature of the hash. VerifySignature only returns a TRUE or FALSE answer. The caller does not receive any information as to the reason for a failure.

The prohibition of returning any error information is especially important for TPMs that implement TSS_VerifySignature as operations on the TPM.

*End of informative comment.*

## 8.8 Random Numbers

***Start of informative comment:***

The TPM has the ability to generate random numbers. This section merely exposes these numbers to allow entities outside of the TPM to use a random number.

The size of the output random area is only limited by the size requested.

Some random number generator implementations are strengthen by adding entropy to the RNG at various intervals. The stir command allows those implementations to receive the entropy when it is available.

***End of informative comment.***

### 8.8.1 TPM_GetRandom

Start of informative comment:

GetRandom returns the next *bytesRequested* bytes from the random number generator to the caller.

End of informative comment.

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetRandom. |
| 4 | 4 | | | UINT32 | bytesRequested | Number of bytes to return |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | UINT32 | randomBytesSize | The number of bytes returned |
| 5 | <> | | | BYTE[ ] | randomBytes | The returned bytes |

**Actions**

1. The TPM determines if amount *bytesRequested* is available from the TPM.
2. Set *randomBytesSize* to the number of bytes available from the RNG. This number MAY be less than *randomBytesSize*.
3. Set *randomBytes* to the next *randomBytesSize* bytes from the RNG
4. It is RECOMMENDED that a TPM implement the RNG in a manner that would allow it to return RNG bytes such that the frequency of *bytesRequested* being less than the number of bytes available be a infrequent occurrence.

### 8.8.2 TPM_StirRandom

Start of informative comment:

StirRandom adds entropy to the RNG state.

End of informative comment.

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_StirRandom |
| 4 | 4 | | | UINT32 | dataSize | Number of bytes of input (<256) |
| 5 | <> | | | BYTE[ ] | inData | Data to add entropy to RNG state |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Actions**

The TPM updates the state of the current RNG using the appropriate mixing function.

## 8.9 Self Test

***Start of informative comment:***

The self-test capabilities are designed to enable the creation of a TCPA platform with minimum latency due to TPM self-test. It might be possible to avoid wasting time, waiting for a TPM to do self-test, by designing a platform where TPM self-testing is done in parallel with other system functions, at a time when TPM capabilities are not required.

At startup, a TPM automatically tests just those internal functions that are used by critical TPM capabilities. This permits the use of those critical TPM capabilities as soon as possible after startup. Remaining TPM capabilities use additional internal functions that must be tested before the remaining TPM capabilities can execute. A test of the additional functions can be explicitly called. Alternatively, those functions will automatically be tested prior to execution of the first call to a capability that uses those functions. At any time, other self-test commands will explicitly cause the TPM to do a full self-test.

TPM_SelfTestFull causes the TPM to do a full self-test.

TPM_CertifySelfTest causes the TPM to do a full self-test and sign the result. It enables the caller to verify that the self-test actually executed and trust the answer. It requires authorization to use a signing key inside the TPM. If the command fails for any reason, the command will not return a signature. The lack of a signature field returning to a Challenger is in itself an indication that some part of the process failed. The failure could be an attack against the signature or a failure in the TPM.

TPM_ContinueSelfTest causes the TPM to test the TPM internal functions that were not tested at startup. TPM_ContinueSelfTest is unusual, in that it returns a result code to the caller before execution of the command and does not return a result code to the caller after execution of the command. If the functions used by a capability have not been tested, TPM_ContinueSelfTest is executed automatically after that capability is called and before it is executed. It is anticipated that the caller or TPM driver software is preprogrammed with knowledge of the time that the TPM will require to complete TPM_ContinueSelfTest. It is anticipated that a call to a TPM that is executing TPM_ContinueSelfTest would result in a "busy" indication.

The tests themselves only return a TCPA_SUCCESS or TCPA_FAIL answer. TPM_GetTestResult must be used to discover why self-test failed. Upon the failure of a self-test the TPM goes into failure mode and does not allow most other operations to continue.

***End of informative comment.***

At startup, a TPM MUST self-test all internal functions that are necessary to do TPM_SHA1Start, TPM_SHA1Update, TPM_SHA1Complete, TPM_SHA1CompleteExtend, TPM_Extend, TPM_Startup, TPM_ContinueSelfTest. This process MUST take 20ms or less.

TSC commands do not operate on shielded locations and have no requirement to be self tested before any use. TPM's SHOULD test these functions before operation.

Some internal functions MUST be tested before the TPM responds to any capability (see 10.8.1). Some internal functions SHOULD be tested before the TPM responds to any capability (see 10.8.2).

If self test has failed, the TPM SHALL respond to all commands (except the update commands) with the error code TCPA_FAILEDSELFTEST (see 10.8.3).

If the functions used by a capability have not been tested, TPM_ContinueSelfTest is executed automatically after that capability is called and before it is executed returning the error TCPA_NEED_SELFTEST

### 8.9.1 TPM_SelfTestFull

Start of informative comment:

SelfTestFull tests all of the TCPA protected capabilities

End of informative comment.

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SelfTestFull |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Actions**

1. TPM_SelfTestFull SHALL cause a TPM to perform self-test of each TPM internal function.
2. Failure of any test results in overall failure, and the TPM goes into failure mode.

### 8.9.2 TPM_CertifySelfTest

*Start of informative comment:*

CertifySelfTest causes the TPM to perform a full self-test and return an authenticated value if the test passes.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have authorization data.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization to use the keyHandle parameter.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CertifySelfTest |
| 4 | *4* | | | TCPA_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can perform digital signatures. |
| 5 | *20* | 2s | *20* | TCPA_NONCE | antiReplay | AnitReplay nonce to prevent replay of messages |
| 6 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | *20* | | | TCPA_AUTHDATA | privAuth | The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CertifySelfTest |
| 4 | *4* | 3s | *4* | UINT32 | sigSize | The length of the returned digital signature |
| 5 | <> | 4s | <> | BYTE[ ] | sig | The resulting digital signature. |
| 6 | *20* | 2 H1 | *20* | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Description**

The key in keyHandle MUST have a KEYUSAGE value of type TPM_KEY_SIGNING or TPM_KEY_LEGACY or TPM_KEY_IDENTITY.

Information returned by TPM_CertifySelfTest MUST NOT aid identification of an individual TPM.

**Actions**

1. The TPM SHALL perform TPM_SelfTestFull. If the test fails the TPM returns the appropriate error code.
2. After successful completion of the self-test the TPM then validates the authorization to use the key pointed to by keyHandle.
3. Create t1 the null terminated string of "Test Passed"
4. The TPM creates m2 the message to sign by concatenating t1 || AntiReplay || ordinal.
5. The TPM signs m2 using the key identified by keyHandle, and returns the signature as sig.

### 8.9.3 TPM_ContinueSelfTest

*Start of informative comment:*

CotinueSelfTest informs the TPM that it may complete the self test of all TPM functions.

*End of informative comment.*

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_ContinueSelfTest |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Actions**

TPM_ContinueSelfTest SHALL cause the TPM to do all self-tests that are outstanding, since startup. It SHALL immediately respond to the caller with a return code. When TPM_ContinueSelfTest finishes execution, it SHALL NOT respond to the caller with a return code.

The TPM SHALL unilaterally execute the functions of TPM_ContinueSelfTest upon receipt of a command that calls a capability-X that uses untested TPM functions. If the self-test fails, the TPM SHALL return the error code TCPA_FAILEDSELFTEST. If the self-test passes, the TPM SHALL execute capability-X.

### 8.9.4 TPM_GetTestResult

**Start of informative comment:**

TPM_GetTestResult provides manufacturer specific information regarding the results of the self test. This command will work when the TPM is in self test failure mode. The reason for allowing this command to operate in the failure mode is to allow TPM manufacturers to obtain diagnostic information.

**End of informative comment.**

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetTestResult |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | UINT32 | outDataSize | The size of the outData area |
| 5 | <> | | | BYTE[] | outData | The outData this is manufacturer specific |

**Actions**

The TPM SHALL respond to this command with a manufacturer specific block of information that describes the result of the latest self test.

The information MUST NOT contain any data that uniquely identifies an individual TPM.

## 8.10 Reset and Clear Operations

*Start of informative comment:*

Reset is the process of clearing all handles and sessions. The reset does not affect PCR values or volatile flag values that are set on TPM initialization. The reset does not affect the SRK or ownership values.

Clear is the process of returning the TPM to factory defaults. The clear commands need protection from unauthorized use and must allow for the possibility of changing Owners. The clear process has authorized commands and mechanisms to not allow the clear operation to occur.

The clear functionality performs the following tasks:

- Delete SRK. The deletion of the SRK includes the destruction of all protected storage areas below the SRK in the hierarchy. The areas below are not destroyed they just have no mechanism to be loaded anymore.
- All TPM volatile and non-volatile data is set to default value except the endorsement key pair. The clear includes the Owner authorization data, so after performing the clear, the TPM has no Owner. The PCR values are undefined after a clear operation.
- The TPM shall returns TCPA_NOSRK until an Owner is set. After the execution of the clear command, the TPM must go through a power cycle to properly set the PCR values.

The Owner has ultimate control of when a clear occurs.

The Owner can perform the TPM_OwnerClear command using the TPM Owner authorization. If the Owner wishes to disable this clear command and require physical access to perform the clear, the Owner can issue the TPM_DisableOwnerClear command.

During the TPM startup processing anyone with physical access to the machine can issue the TPM_ForceClear command. This command performs the clear. The TPM_DisableForceClear disables the TPM_ForceClear command for the duration of the power cycle. TSS startup code that does not issue the TPM_DisableForceClear leaves the TPM vulnerable to a denial of service attack. The assumption is that the TSS startup code will issue the TPM_DisableForceClear on each power cycle after the TSS determines that it will not be necessary to issue the TPM_ForceClear command. The purpose of the TPM_ForceClear command is to recover from the state where the Owner has lost or forgotten the TPM Ownership token.

The TPM_ForceClear must only be possible when the issuer has physical access to the platform. The manufacturer of a platform determines the exact definition of physical access.

*End of informative comment.*

The TPM MUST support the reset operation. The reset operation clears all handles, authorization sessions and volatile state machines. The reset MUST NOT affect the SRK, PCR and flags such as the flag set by TPM_DisableForceClear.

The TPM MUST support the clear operations. The clear operation MUST perform the following actions:

- Perform a reset operation
- Delete the SRK
- Reset all non-volatile values to factory default except the endorsement key pair
- Return TCPA_NOSRK until there is a proper execution of the ownership function

The TPM MUST support disabling the clear operations. After execution of the TPM_DisableOwnerClear the TPM MUST require physical access to execute the TPM_ForceClear. The TPM MUST support the TPM_DisableForceClear to disable the TPM_ForceClear command. The TPM_DisableForceClear command MUST execute on each startup cycle to be effective.

### 8.10.1 TPM_Reset

Start of informative comment:

TPM_reset releases all resources associated with existing authorisation sessions. This is useful if a TSS driver has lost track of the authorisation state in the TPM, for example.

End of informative comment.

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Reset. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Actions**

1. The TPM frees all resources allocated to authorization sessions extant in the TPM
2. The TPM does not reset any PCR or DIR values.
3. The TPM does not reset any flags in the TCPA_VOLATILE_FLAGS structure.
4. The TPM does not reset or delete any keys

### 8.10.2 TPM_Init

*Start of informative comment:*

TPM_Init is a physical method of initializing a TPM. It calls TPM_reset to release any authorization sessions, and then puts the TPM into a state where it waits for the command TPM_startup (which specifies the type of initialization that is required).

*End of informative comment.*

**Definition**

```
TPM_Init();
```

**Type**

TCPA protected capability that requires physical indication from the platform

**Parameters**

None

**Description**

The platform MUST be designed such that if the TPM_Init signal is asserted the entire Platform MUST be initialized. This prevents, at least with a minimum effort, someone touching the TPM_Init pin on the TPM and resetting only the TPM.

The TPM_Init signal MUST have signaling qualifications appropriate for the required conformance and Protection Profile for the Platform.

**Actions**

1. The TPM performs a TPM_Reset.
2. The TPM sets TCPA_VOLATILE_FLAGS -> postInitialise to TRUE. See 4.13.3 for details of the "postInitialise" state.

### 8.10.3 TPM_SaveState

*Start of informative comment:*

This warns a TPM to save some state information.

If the relevant shielded storage is non-volatile, this command need have no effect.

If the relevant shielded storage is volatile and the TPM alone is unable to detect the loss of external power in time to move data to non-volatile memory, this command should be presented before the TPM enters a low or no power state.

*End of informative comment.*

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SaveState. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Description**

Preserved values MUST be non-volatile.

If data is never stored in a volatile medium, that data MAY be used as preserved data. In such cases, no explicit action may be required to preserve that data.

If an explicit action is required to preserve data, it MUST be possible to determine whether preserved data is valid.

If the parameter mirrored by a preserved value is altered, the preserved value MUST be declared invalid. If the parameter mirrored by any preserved value is altered, all preserved values MAY be declared invalid.

**Actions**

1. The contents of all PCRs MUST be preserved.
2. The contents of the auditDigest MUST be preserved.
3. The state of the flags:
    i. TCPA_VOLATILE_FLAGS -> PhysicalPresence
    ii. TCPA_VOLATILE_FLAGS -> PhysicalPresenceLock

iii. TCPA_VOLATILE_FLAGS -> deactivated

iv. TCPA_VOLATILE_FLAGS -> disableForceClear

MUST be preserved.

4. The contents of any key that is currently loaded SHOULD be preserved if the key's parentPCRStatus indicator is FALSE and its IsVolatile indicator is FALSE. The contents of any key that is currently loaded MAY be preserved if its parentPCRStatus indicator is TRUE or its IsVolatile indicator is TRUE.

## 8.10.4 TPM_Startup

*Start of informative comment:*

Some trusted entity must determine the type of startup state that is required and submit TPM_Startup with the appropriate option.

TPM_Startup must always be preceded by TPM_Init, which is a physical indication (probably just a system-wide reset signal) to a TPM that initialization is required. Determining the type of initialization requires more intelligence than may be available from a simple physical mechanism, so TPM_Startup is used to signal the type of initialization that is required.

A key that is itself wrapped to PCRs is not unloaded at startup because:

a) existing mechanisms (specified in TPM_LoadKey) prevent use of the key unless the PCRs match. So it is unnecessary to unload the key.

b) the key may be required for later use, without reloading, in which case it is undesirable to unload the key.

*End of informative comment.*

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Startup |
| 4 | 2 | | | TCPA_STARTUP_TYPE | startupType | Type of startup that is occurring |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Description**

TPM_Startup MUST be generated by a trusted entity (the RTM or the TPM, for example).

**Actions**

1. If no EK is present, the TPM MUST return TCPA_NO_ENDORSEMENT and exit this capability.
2. If TCPA_VOLATILE_FLAGS -> postInitialise is FALSE, the TPM MUST return TCPA_INVALID_POSTINIT, and exit this capability.
3. **If stType = TCPA_ST_CLEAR**
    a. Reset PCR's

b. Reset the auditDigest

c. The TPM Must set the following flags to their default state:

   i. TCPA_VOLATILE_FLAGS -> PhysicalPresence

   ii. TCPA_VOLATILE_FLAGS -> PhysicalPresenceLock

   iii. TCPA_VOLATILE_FLAGS -> disableForceClear

d. The TPM SHALL set TCPA_VOLATILE_FLAGS -> deactivated to the same state as TCPA_PERSISTENT_FLAGS -> deactivated

e. The TPM SHALL take all necessary actions to ensure that all loaded keys contain the preserved value if the preserved value is valid and the preserved value's parentPCRStatus indicator is FALSE and its IsVolatile indicator is FALSE. All other key areas MUST be unloaded. If the TPM is unable to successfully complete these actions, it SHALL enter the TPM failure mode.

4. **If stType = TCPA_ST_STATE**

   a. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid preserved values. If the TPM is unable to successfully complete these actions, it SHALL enter the TPM failure mode.

   b. The TPM SHALL take all necessary actions to ensure that the auditDigest contains a valid preserved value. If the TPM is unable to successfully complete these actions, it SHALL enter the TPM failure mode.

   c. The TPM MUST restore the following flags to their preserved states:

      i. TCPA_VOLATILE_FLAGS -> PhysicalPresence

      ii. TCPA_VOLATILE_FLAGS -> PhysicalPresenceLock

      iii. TCPA_VOLATILE_FLAGS -> deactivated

      iv. TCPA_VOLATILE_FLAGS -> disableForceClear

   d. The TPM MUST restore all keys that have been saved

   e. The TPM resumes normal operation. If the TPM is unable to resume normal operation, it SHALL enter the TPM failure mode.

5. **If stType = TCPA_ST_DEACTIVATED**

   a. The TPM MUST set TCPA_VOLATILE_FLAGS -> deactivated to TRUE

6. The TPM MUST invalidate any explicitly preserved state and set TCPA_VOLATILE_FLAGS -> postInitialise to FALSE.

### 8.10.5 TPM_OwnerClear

*Start of informative comment:*

The OwnerClear command performs the clear operation under Owner authorization. This command is available until the Owner executes the DisableOwnerClear, at which time any further invocation of this command returns TCPA_CLEAR_DISABLED.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization as the TPM Owner.

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerClear |
| 4 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Ignored |
| 7 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2S | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerClear |
| 4 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Fixed value FALSE |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: old ownerAuth. |

**Actions**

1. The TPM verifies that the authHandle properly authorizes the owner.
2. After owner verification the TPM then checks the status of the TCPA_PERSISTENT_FLAGS -> DisableOwnerClear flag, if set the TPM returns TCPA_CLEAR_DISABLED.

3. The TPM executes the TPM_Reset command. The TPM then destroys the SRK and any internal data associated with the SRK. The TPM then destroys the TPM Ownership data.
4. The TPM unloads all loaded keys.
5. The TPM sets all DIR registers to their default value.
6. The TPM sets TCPA_PERSISTENT_FLAGS to their default values.
7. The result will be no Owner or SRK and the TPM is set to the state where it returns TCPA_NOSRK.

### 8.10.6 TPM_DisableOwnerClear

Start of informative comment:

The DisableOwnerClear command disables the ability to execute the TPM_OwnerClear command permanently. Once invoked the only method of clearing the TPM will require physical access to the TPM.

End of informative comment.

**Type**

TCPA protected capability; user must provide authorization as the TPM Owner.

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisableOwnerClear |
| 4 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisableOwnerClear |
| 4 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. The TPM verifies that the authHandle properly authorizes the owner.
2. The TPM sets the TCPA_PERSISTENT_FLAGS -> disableownerclear flag to TRUE.
3. The only mechanism that can clear the TPM is the TPM_ForceClear command. The TPM_ForceClear command requires physical access to the TPM to execute.

### 8.10.7 TPM_ForceClear

*Start of informative comment:*

The ForceClear command performs the Clear operation under physical access. This command is available until the execution of the DisableForceClear, at which time any further invocation of this command returns TCPA_CLEAR_DISABLED.

*End of informative comment.*

**Type**

TCPA protected capability; there must be some evidence of physical access to the platform present for the TPM to verify.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_ForceClear |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Actions**

1. The TPM checks for a prior execution of the TPM_DisableForceClear command. If executed, the TPM will return TCPA_CLEAR_DISABLED.
2. After verification of physical access, the TPM performs a clear operation that has the same result as the TPM_OwnerClear. After execution the result of this command is exactly like the TPM_OwnerClear.
3. The implementation of the physical access requirement is a manufacturer option. The evidence of physical access could be done by setting a pin high on a chip, or by sending special bus cycles or by any other mechanism that provides evidence of physical access.

### 8.10.8 TPM_DisableForceClear

*Start of informative comment:*

The DisableForceClear command disables the execution of the ForceClear command until the nex startup cycle. Once this command is executed the TPM_ForceClear is disabled until another startup cycle is run.

*End of informative comment.*

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_DisableForceClear |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Actions**

The TPM sets the TCPA_VOLATILE_FLAGS.disableforceclear flag in the TPM that disables the execution of the TPM_ForceClear command.

## 8.11 The GetCapability Commands

*Start of informative comment:*

The TPM has numerous capabilities that a remote entity may wish to know about. These items include support of algorithms, key sizes, protocols and vendor-specific additions. The GetCapability command allows the TPM to report back to the requestor what type of TPM it is dealing with.

There are two variations of the GetCapability command: one that provides a signed response and one that merely returns the answer without an accompanying signature. The information in each is the same except for the inclusion or absence of a digital signature.

The request for information requires the requestor to specify which piece of information that is required. The request does not allow the "merging" of multiple requests and returns only a single piece of information.

In failure mode the TPM can only return manufacturer's name, TPM model and TPM version.

*End of informative comment.*

The TPM MUST NOT return in response to the GetCapability command any information that identifies an individual TPM.

### 8.11.1 TPM_GetCapability

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapability |
| 4 | 4 | | | TCPA_CAPABILITY_AREA | capArea | Partition of capabilities to be interrogated |
| 5 | 4 | | | UINT32 | subCapSize | Size of subCap parameter |
| 6 | <> | | | BYTE[] | subCap | Further definition of information |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | UINT32 | respSize | The length of the returned capability response |
| 5 | <> | | | BYTE[ ] | resp | The capability response |

**Actions**

The TPM validates the capArea and subCap indicators. If the information is available, the TPM creates the response field and fills in the actual information.

| CapArea | subCap | Response |
|---|---|---|
| TCPA_CAP_ORD | `ORDINAL:` `A value of command ordinal:` `see 4.32` | Boolean value. TRUE indicates that the TPM supports the ordinal. FALSE indicates that the TPM does not support the ordinal. |
| TCPA_CAP_ALG | `TCPA_ALG_XX:` `A value of TCPA_ALGORITHM_ID:` `see 4.15` | Boolean value. TRUE indicates that the TPM supports the algorithm, FALSE indicates that the TPM does not support the algorithm. |
| TCPA_CAP_PID | `TCPA_PID:` `A value of TCPA_PROTOCOL_ID:` `See 4.15` | Boolean value. TRUE indicates that the TPM supports the protocol, FALSE indicates that the TPM does not support the protocol. |
| TCPA_CAP_PROPERTY | `TPM_CAP_PROP_PCR` | UINT32 value. Returns the number of PCR registers supported by the |

| | | |
|---|---|---|
| | | of PCR registers supported by the TPM |
| TCPA_CAP_PROPERTY | TPM_CAP_PROP_DIR | UINT32 value. Returns the number of DIR registers supported by the TPM. |
| TCPA_CAP_PROPERTY | TCPA_CAP_PROP_MANUFACTURER | UINT32 value. Returns the Identifier of the TPM manufacturer. |
| TCPA_CAP_PROPERTY | TCPA_CAP_PROP_SLOTS | UINT32 value. Returns the maximum number of 2048 bit RSA keys that the TPM is capable of loading. This MAY vary with time and circumstances. |
| TCPA_CAP_VERSION | Ignored | Returns the TCPA_VERSION structure that identifies the version of the TPM. See 4.5 |
| TCPA_CAP_KEY_HANDLE | Ignored | A TCPA_KEY_HANDLE_LIST structure, describing the handles of all keys that are currently loaded into the TPM. See 4.9 |
| TCPA_CAP_CHECK_LOADED | ALGORITHM: A value of TCPA_KEY_PARMS: see 4.15 | A Boolean value. TRUE indicates that the TPM has enough memory available to load a key of the type specified by ALGORITHM. FALSE indicates that the TPM does not have enough memory. |

The permitted values of TCPA_CAP_PROP_MANUFACTURER and their meaning SHALL be defined in platform specific TCPA specifications.

**IDL Definitions of subCap**

```
#define TCPA_CAP_PROP_PCR              0x00000101
#define TCPA_CAP_PROP_DIR              0x00000102
#define TCPA_CAP_PROP_MANUFACTURER     0x00000103
#define TCPA_CAP_PROP_SLOTS            0x00000104
```

### 8.11.2 TPM_GetCapabilitySigned

*Start of informative comment:*

TPM_GetCapabilitySigned is almost the same as TPM_GetCapability. The differences are that the input includes a challenge (a nonce) and the response includes a digital signature to vouch for the source of the answer.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have authorization data.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

*End of informative comment.*

**Type**

TCPA protected capability; the user must supply authorization to use of parameter keyHandle

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapabilitySigned |
| 4 | 4 | | | TCPA_KEY_HANDLE | keyHandle | The handle of a loaded key that can perform digital signatures. |
| 5 | 20 | 2s | 20 | TCPA_NONCE | antiReplay | Nonce provided to allow caller to defend against replay of messages |
| 6 | 4 | 3s | 4 | TCPA_CAPABILITY_AREA | capArea | Partition of capabilities to be interrogated |
| 7 | 4 | 4s | 4 | UINT32 | subCapSize | Size of subCap parameter |
| 8 | <> | 5s | <> | BYTE[] | subCap | Further definition of information |
| 8 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TCPA_AUTHDATA | privAuth | The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapabilitySigned |
| 4 | 4 | 3s | 4 | TCPA_VERSION | version | A properly filled out version structure. |
| 5 | 4 | 4s | 4 | UINT32 | respSize | The length of the returned capability response |
| 6 | <> | 5s | <> | BYTE[] | resp | The capability response |
| 7 | 4 | 6s | 4 | UINT32 | sigSize | The length of the returned digital signature |
| 8 | <> | 7s | <> | BYTE[] | sig | The resulting digital signature. |
| 9 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 11 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

### Description

The key in keyHandle MUST have a KEYUSAGE value of type TPM_KEY_SIGNING or TPM_KEY_LEGACY or TPM_KEY_IDENTITY.

### Actions

1. The TPM calls TPM_GetCapability passing the capArea and subCap fields and saving the resp field as r1.
2. The TPM creates h1 by taking a SHA1 hash of the concatenation (r1 || antiReplay).
3. The TPM validates the authority to use keyHandle
4. The TPM creates a digital signature of h1 using the key in keyHandle and returns the result in sig.

### 8.11.3 TPM_GetCapabilityOwner

*Start of informative comment:*

TPM_GetCapabilityOwner enables the TPM Owner to retrieve all the non-volatile flags and the volatile flags in a single operation.

The flags summarize many operational aspects of the TPM. The information represented by some flags is private to the TPM Owner. So, for simplicity, proof of ownership of the TPM must be presented to retrieve the set of flags. When necessary, the flags that are not private to the Owner can be deduced by Users via other (more specific) means.

The normal TCPA authorization mechanisms are sufficient to prove the integrity of the response. No additional integrity check is required.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authentication from the TPM Owner.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapbilityOwner |
| 3 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for Owner authorization. |
| | | 2 H1 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 4 | *20* | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 6 | *20* | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: OwnerAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | *4* | 1s | *4* | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 3 | *4* | 2s | *4* | TCPA_VERSION | version | A properly filled out version structure. |
| 4 | *4* | 3s | *4* | UINT32 | non_volatile_flags | The current state of the non-volatile flags. |
| 5 | *4* | 4s | *4* | UINT32 | volatile_flags | The current state of the volatile flags. |
| 6 | *20* | 2 H1 | *20* | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | *1* | 4 H1 | *1* | BOOL | continueAuthSession | Continue use flag. TRUE if handle is still active |
| 8 | *20* | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | HMAC key: OwnerAuth. |

**Description**

For 31>=N>=0

- Bit-N of the TCPA_PERSISTENT_FLAGS structure is the Nth bit after the opening bracket in the definition of TCPA_PERSISTENT_FLAGS in the version of the specification indicated by the parameter "version". The bit immediately after the opening bracket is the $0^{th}$ bit.
- Bit-N of the TCPA_VOLATILE_FLAGS structure is the Nth bit after the opening bracket in the definition of TCPA_VOLATILE_FLAGS in the version of the specification indicated by the parameter "version". The bit immediately after the opening bracket is the $0^{th}$ bit.
- Bit-N of non_volatile_flags corresponds to the Nth bit in TCPA_PERSISTENT_FLAGS.
- Bit-N of volatile_flags corresponds to the Nth bit in TCPA_VOLATILE_FLAGS.

**Actions**

1. The TPM validates that the TPM Owner authorizes the command.
2. The TPM creates the parameter non_volatile_flags by setting each bit to the same state as the corresponding bit in TCPA_PERSISTENT_FLAGS. Bits in non_volatile_flags for which there is no corresponding bit in TCPA_PERSISTENT_FLAGS are set to zero.
3. The TPM creates the parameter volatile_flags by setting each bit to the same state as the corresponding bit in TCPA_VOLATILE_FLAGS. Bits in volatile_flags for which there is no corresponding bit in TCPA_VOLATILE_FLAGS are set to zero.
4. The TPM generates the parameter "version".
5. The TPM returns non_volatile_flags, volatile_flags and version to the caller.

## 8.12 Audit Commands

*Start of informative comment:*

The TPM and TSS need to be able to report a log of events. The log uses the same paradigm as the PCRs: the TPM keeps a PCR value that extends for each log event, and the TSS maintains the log entries for Challengers to review.

The Owner has the ability to set which functions generate an audit event and to change which functions generate the event at any time.

The status of the audit generation is not seen as sensitive information and so the command to determine the status of the generation is not an authorized command.

*End of informative comment.*

Each command ordinal has an indicator in non-volatile TPM memory indicating if executing the command will result in the generation of an audit event.

The audit event includes the command ordinal and the return code from the command.

The digest value SHALL be SHA1 (previous value || command ordinal || return code). The digest value register SHALL have a starting value of NULLS.

Updating of auditDigest MAY cease when TCPA_VOLATILE_FLAGS -> deactivated is TRUE. This is because a deactivated TPM performs no useful service until a platform is rebooted, at which point auditDigest is reset.

### 8.12.1 TPM_GetAuditEvent

Start of informative comment:

The TPM uses this command to get the audit information from the TPM.

End of informative comment.

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetAuditEvent |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | TCPA_COMMAND_CODE | cmdOrd | Last audited command executed |
| 5 | 4 | | | UINT32 | cmdReturnCode | Return code for cmdOrd |
| 6 | 20 | | | TCPA_DIGEST | auditDigest | Log of all audited events |

**Actions**

1. The TPM sets cmdOrd to the ordinal of the last audited function.
2. The TPM sets cmdReturnCode to the return code for the last audited function.
3. The TPM sets auditDigest to the extended digest value of all audited functions.

## 8.12.2 TPM_GetAuditEventSigned

Start of informative comment:

This command returns the same information as the TPM_GetAuditEvent but the result is signed.

End of informative comment.

**Type**

TCPA protected capability; user must provide authentication to use the key pointed to by keyHandle.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetAuditEventSigned |
| 4 | 4 | | | TCPA_KEY_HANDLE | keyHandle | The handle of a loaded key that can perform digital signatures. |
| 5 | 20 | 2s | 20 | TCPA_NONCE | antiReplay | A nonce to prevent antiReplay attacks |
| 6 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for key authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TCPA_AUTHDATA | keyAuth | The authorization digest for inputs and owner authorization. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetAuditEventSigned |
| 4 | 4 | 3s | 4 | TCPA_COMMAND_CODE | cmdOrd | Last audited command executed |
| 5 | 4 | 4s | 4 | UINT32 | cmdReturnCode | Return code for cmdOrd |
| 6 | 20 | 5s | 20 | TCPA_DIGEST | auditDigest | Log of all audited events |
| 7 | 4 | 6s | 4 | UINT32 | ordSize | The size of the ordinal list |
| 8 | <> | 7s | <> | BYTE[] | ordinalList | The list of ordinals that are being audited |
| 9 | 4 | 8s | 4 | UINT32 | sigSize | The size of the sig parameter |
| 10 | <> | 9s | <> | BYTE[] | sig | The signature of the area |
| 11 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 13 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth. |

**Actions**

1. The TPM sets cmdOrd to the ordinal of the last audited function.
2. The TPM sets cmdReturnCode to the return code for the last audited function.
3. The TPM sets auditDigest to the extended digest value of all audited functions.
4. The TPM sets ordinalList to a list of all audited functions. This list is a UINT32 of command ordinals.
5. Create a d1 by taking the SHA1 of (ordinal || cmdOrd || cmdReturnCode || auditDigest || ordinalList || antiReplay)
6. Create a digital signature of d1 by using the signature scheme for keyHandle.
7. Return the signature in the sig parameter

### 8.12.3 TPM_SetOrdinalAuditStatus

Start of informative comment:

Set the audit flag for a given ordinal. This command requires the authorization of the TPM Owner.

End of informative comment.

**Type**

TCPA protected capability; the user must show authorization from the TPM Owner to execute the command.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOrdinalAuditStatus |
| 4 | 4 | 2s | 4 | TCPA_COMMAND_CODE | ordinalToAudit | The ordinal whose audit flag is to be set |
| 5 | 1 | 3s | 1 | BOOL | auditState | Value for audit flag |
| 6 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOrdinalAuditStatus |
| 4 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Descriptions**

**Actions**

1. The TPM authenticates the command using the TPM Owner authentication. If authentication unsuccessful the TPM returns TCPA_FAIL.

2. The TPM sets the state of the non-volatile flag for the given ordinal to the indicated state. The TPM also returns the state in the response.

### 8.12.4 TPM_GetOrdinalAuditStatus

Start of informative comment:

Get the status of the audit flag for the given ordinal.

End of informative comment.

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetOrdinalAuditStatus |
| 4 | 4 | | | TCPA_COMMAND_CODE | ordinalToQuery | The ordinal whose audit flag is to be queried |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 1 | | | BOOL | State | Value of audit flag for ordinalToQuery |

**Actions**

The TPM returns the Boolean value for the given ordinal. The value is TRUE if the command is being audited.

### 8.12.5 Effect of audit failing after successful completion of a command

*Start of informative comment:*

An operation could complete successfully and then when the TPM attempts to audit the command the audit process could have an internal error that forces the TPM to return an error.

This section indicates what the TPM must do in this case in addition to setting the state that requires the TPM to return TPM_FAILEDSELFTEST.

*End of informative comment.*

When after successful completion of an operation, and in performing the audit process, the TPM has an internal failure (unable to write, SHA failure etc.) the TPM MUST set the internal TPM state such that the TPM returns the TPM_FAILEDSELFTEST error. The TPM MUST return TCPA_AUDITFAILURE for the current command.

If the TPM is permanently nonrecoverable after an audit failure, then the TPM MUST always return TPM_FAILEDSELFTEST for every command other than TPM_GetTestResult. This state must persist regardless of power cycling, the execution of TPM_Init or any other actions.

If the TPM can recover in any way after the failure of an audit operation, then the TPM MUST take the actions stated in the following table after setting the failure state.

| Ordinal | Effect when Audit Fails |
|---|---|
| TPM_ORD_OIAP | No action - session deleted on TPM_INIT |
| TPM_ORD_OSAP | No action - session deleted on TPM_INIT |
| TPM_ORD_ChangeAuth | No action - changed blob not returned so nothing to delete |
| TPM_ORD_TakeOwnership | TPM returns to state where there is no TPM Owner. |
| TPM_ORD_ChangeAuthAsymStart | No action - session deleted on TPM_INIT |
| TPM_ORD_ChangeAuthAsymFinish | No action - session deleted on TPM_INIT |
| TPM_ORD_ChangeAuthOwner | The TPM MUST revert back to the previous authorization value |
| | |
| TPM_ORD_Extend | Invalidate PCR by extending 20 bytes of 0xa5 to the PCR |
| TPM_ORD_PcrRead | No action |
| TPM_ORD_Quote | No action |
| TPM_ORD_Seal | No action |
| TPM_ORD_Unseal | Ensure that unsealed data is made unavailable |
| TPM_ORD_DirWriteAuth | Invalidate the DIR by writing 20 bytes of 0xa5 into the specified DIR |
| TPM_ORD_DirRead | No action |
| | |
| TPM_ORD_UnBind | Ensure that unbound data is made unavailable |
| TPM_ORD_CreateWrapKey | No action - key not returned in blob so TPM can just lose the new key |
| TPM_ORD_LoadKey | Ensure that the key is not available |
| TPM_ORD_GetPubKey | No action - nothing returned |
| TPM_ORD_EvictKey | No action - key is evicted so no security issues |
| | |
| TPM_ORD_CreateMigrationBlob | No action - no blob returned |

| | |
|---|---|
| TPM_ORD_ReWrapKey | No action - no blob returned |
| TPM_ORD_ConvertMigrationBlob | No action - no blob returned |
| TPM_ORD_AuthorizeMigrationKey | No action - no blob returned |
| TPM_ORD_CreateMaintenanceArchive | No action - no blob returned |
| TPM_ORD_LoadMaintenanceArchive | Set the TPM internal state such that the TPM returns TPM_NOSRK. This requires the caller to resubmit the maintenance archive for it to be active. |
| TPM_ORD_KillMaintenanceFeature | No action |
| TPM_ORD_LoadManuMaintPub | The TPM returns to a state where no maintenance public key has been loaded |
| TPM_ORD_ReadManuMaintPub | No action - no blob returned |
| | |
| TPM_ORD_CertifyKey | No action - no blob returned |
| | |
| TPM_ORD_Sign | No action - no blob returned |
| | |
| TPM_ORD_GetRandom | No action - nothing returned |
| TPM_ORD_StirRandom | No action - RNG still secure |
| | |
| TPM_ORD_SelfTestFull | No action |
| TPM_ORD_SelfTestStartup | No action |
| TPM_ORD_CertifySelfTest | No action |
| TPM_ORD_ContinueSelfTest | No action |
| TPM_ORD_GetTestResult | No action |
| | |
| TPM_ORD_Reset | No action |
| TPM_ORD_OwnerClear | No action |
| TPM_ORD_DisableOwnerClear | No action |
| TPM_ORD_ForceClear | No action |
| TPM_ORD_DisableForceClear | No action |
| | |
| TPM_ORD_GetCapabilitySigned | No action |
| TPM_ORD_GetCapability | No action |
| TPM_ORD_GetCapabilityOwner | No action |
| | |
| TPM_ORD_OwnerSetDisable | No action |
| TPM_ORD_PhysicalEnable | No action |
| TPM_ORD_PhysicalDisable | No action |
| TPM_ORD_SetOwnerInstall | No action |
| TPM_ORD_PhysicalSetDeactivated | No action |
| TPM_ORD_SetTempDeactivated | No action |
| | |
| TPM_ORD_CreateEndorsementKeyPair | This is a dead TPM. It has failed it's startup smoke test. It should not leave the factory floor. |
| TPM_ORD_MakeIdentity | No action - blob not returned so key is lost |
| TPM_ORD_ActivateIdentity | No action - credential not returned but blob is still available for the caller to resubmit to the TPM when it is functional |
| TPM_ORD_ReadPubek | No action |
| TPM_ORD_OwnerReadPubek | No action |
| TPM_ORD_DisablePubekRead | No action |

| | |
|---|---|
| | |
| TPM_ORD_GetAuditEvent | No action |
| TPM_ORD_GetAuditEventSigned | No action |
| | |
| TPM_ORD_GetOrdinalAuditStatus | No action |
| TPM_ORD_SetOrdinalAuditStatus | No action |
| | |
| TPM_ORD_Terminate_Handle | No action |
| TPM_ORD_Init | No action |
| TPM_ORD_SaveState | No action |
| TPM_ORD_Startup | No action - The TPM is disabled, all save states are invalidated so only non-volatile keys are left. |
| TPM_ORD_SetRedirection | No action |
| | |
| TPM_ORD_SHA1Start | No action |
| TPM_ORD_SHA1Update | No action |
| TPM_ORD_SHA1Complete | No action |
| TPM_ORD_SHA1CompleteExtend | No action |
| | |
| TPM_ORD_FieldUpgrade | Set TCPA_PERSISTENT_FLAGS -> FailedFieldUpgrade to TRUE. This flag sets the disabled bit to TRUE on each TPM_Init. The only way to set the FailedFieldUpgrade flag back to FALSE is to successfully complete a FieldUpgrade. |

## 8.13 Enabling Ownership

*Informative comment*

The purpose of these capabilities is to enable and disable the process of taking ownership of a TPM.

The process of enabling and disabling ownership uses a non-volatile flag TCPA_PERSISTENT_FLAGS-> ownership. If the TCPA_PERSISTENT_FLAGS -> ownership flag is FALSE, the TPM will not permit the "take ownership" command to operate. If the flag is TRUE, it has no effect on any other capability. See section 4.13.1 for the TCPA_PERSISTENT_FLAGS -> ownership flag.

This enableOwnership command on its own does not provide the necessary privacy controls for a TPM. It should be considered together with the operation of the enable/disable commands of section 8.14, and the activate/deactivate commands of section 8.15. The activate/deactivate commands are weaker forms of the enable/disable commands, in that they permit the process of taking Ownership of a TPM. The enableOwnership, enable/disable, and activate/deactivate commands together permit the taking of TPM Ownership without the risk of inadvertent use of a TPM. See section 2.6.

Physical presence authorizes the changing of the TCPA_PERSISTENT_FLAGS -> ownership flag.

A remote entity must not be able to change the setting of the TCPA_PERSISTENT_FLAGS -> ownership flag without the collusion of someone present at the platform.

*End of informative comment*

### 8.13.1 TPM_SetOwnerInstall

**Type**

TCPA protected capability; there must be some evidence of physical access present for the TPM to verify.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOwnerInstall |
| 4 | 1 | | | BOOL | state | State to which ownership flag is to be set. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Action**

1. If the TPM has a current owner, this command immediately returns with TCPA_SUCCESS.
2. The TPM validates the assertion of physical access. The TPM then sets the value of TCPA_PERSISTENT_FLAGS -> ownership to the value in state.

## 8.14 Enabling a TPM

***Informative comment***

The purpose of these capabilities is to enable and disable a TPM without destroying secrets protected by the TPM.

The process of enabling and disabling a TPM uses the non-volatile TCPA_PERSISTENT_FLAGS disable flag. When set to TRUE, the TPM will reject most commands. Note, however, that a disabled TPM never disables the "extend" capability. This is necessary in order to ensure that the PCR values in a TPM are always up-to-date. If the flag is FALSE, it has no effect on other capabilities. See section 4.13.1 for the full effects of the TCPA_PERSISTENT_FLAGS disable flag.

These enable/disable commands on their own do not provide the necessary privacy controls for a TPM. They should be considered together with the operation of the enable-ownership command of section 8.12.5 and the activate/deactivate commands of section 8.15. The activate/deactivate commands are weaker forms of the enable/disable commands, in that they permit the process of taking Ownership of a TPM. The enable-Ownership, enable/disable, and activate/deactivate commands together permit the taking of TPM Ownership without the risk of inadvertent use of a TPM. See section 2.6.

There are two mechanisms to change the status of the TCPA_PERSISTENT_FLAGS disable flag. The first mechanism is by using the owner-authenticated command TPM_OwnerSetDisable. The second uses the two commands TPM_PhysicalEnable and TPM_PhysicalDisable. These two commands require the assertion of physical presence. TPM_PhysicalEnable must be incapable of subversion by software.

***End of informative comment***

### 8.14.1 TPM_OwnerSetDisable

**Type**

TCPA protected capability; the TPM Owner must provide authorization.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerSetDisable |
| 4 | 1 | 2s | 1 | BOOL | disableState | Value for disable state – enable if TRUE |
| 5 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerSetDisable |
| 4 | 20 | 2H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Action**

1. The TPM SHALL authenticate the command as coming from the TPM Owner. If unsuccessful, the TPM SHALL return TCPA_BAD_AUTH.
2. The TPM SHALL set the TCPA_PERSISTENT_FLAGS -> disable flag to the value in the disableState parameter.

### 8.14.2 TPM_PhysicalDisable

**Type**

TCPA protected capability; there must be some evidence of physical access present for the TPM to verify.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_PhysicalDisable |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Action**

The TPM SHALL set the TCPA_PERSISTENT_FLAGS.disable value to TRUE. The TPM while executing this command MUST obtain assurance from a physical method that operation of this command is authorized.

The TPM manufacturer MAY implement this command not as a response to a message block but as a response to a physical action, for instance, the acceptance of a special bus cycle or setting a pin high.

## 8.14.3 TPM_PhysicalEnable

### Type

TCPA protected capability; there MUST be unambiguous evidence of the presence of physical access to the platform for the TPM to verify.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_PhysicalEnablel |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

### Action

The TPM SHALL set the TCPA_PERSISTENT_FLAGS.disable value to FALSE.

In order to execute this command, the TPM MUST obtain unambiguous assurance that operation of this command is authorized by physical presence at the platform. The command MAY be initiated by the presentation to a TPM of a message block with the above input parameters, provided that the message block occurs while the TPM is presented with unambiguous assurance that operation of this command is authorized by physical presence at the platform.

Unambiguous assurance that operation of this command is authorized by a physical action at the platform MAY be communicated to a TPM using a special bus cycle that is impossible for software to create, or asserting a single electrical signal that is impossible for software to create, for example.

It SHALL be impossible to subvert this command to a TPM by the execution of instructions in a computing engine on the platform.

## 8.15 Activating a TPM

*Informative comment*

The purpose of these capabilities is to activate and deactivate a TPM without destroying secrets protected by the TPM. This is subtly different from enabling and disabling a TPM.

An inactive TPM permits more commands to operate than does a disabled TPM. In particular, an inactive TPM does not block the enabling/disabling of a TPM and the process of taking ownership of the TPM. An inactive TPM never prevents the "extend" capability from operating. This is necessary in order to ensure that the PCR values in a TPM are always up-to-date.

These activate/deactivate commands on their own do not provide the necessary privacy controls for a TPM. They should be considered together with the operation of the enable-Ownership commands of section 8.12.5 and the enable/disable commands of section 8.14. The enable/disable commands are stronger forms of the activate/deactivate commands, in that they do not permit the process of taking Ownership of a TPM. The enable-Ownership, enable/disable, and activate/deactivate commands together permit the taking of TPM-Ownership without the risk of inadvertent use of a TPM. See section 2.6.

There are TWO deactivated flags: one volatile and one non-volatile. At switch-on, the volatile flag is set to the same state as the non-volatile flag. Altering the non-volatile flag requires physical presence at the platform. The volatile flag can be set without authentication, but its effect lasts only until the platform is rebooted.

See section 4.13.1 for the full effect of the TCPA_PERSISTENT_FLAGS.deactivated flag. See section 4.13.3 for the full effects of the TCPA_VOLATILE_FLAGS.deactivated flag.

*End of informative comment.*

### 8.15.1 TPM_PhysicalSetDeactivated

**Type**

TCPA protected capability; there must be some evidence of physical access present for the TPM to verify.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_PhysicalSetDeactivated |
| 4 | 1 | | | BOOL | state | State to which deactivated flag is to be set. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Action**

The TPM while executing this command MUST obtain assurance from a physical method that operation of this command is authorized.

The TPM SHALL set the TCPA_PERSISTENT_FLAGS.deactivated flag to the value in the state parameter.

### 8.15.2 TPM_SetTempDeactivated

**Type**

TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetTempDeactivated |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |

**Action**

The TPM SHALL set the TCPA_VOLATILE_FLAGS.deactivated flag to the value TRUE.

## 8.16 TPM_FieldUpgrade

*Start of informative comment:*

The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is in the field. Given the varied nature of TPM implementations there will be numerous methods of performing an upgrade of the protected capabilities. This command, when implemented, provides a manufacturer specific method of performing the upgrade.

The manufacturer can determine, within the listed requirements, how to implement this command. The command may be more than one command and actually a series of commands.

The IDL definition is to create an ordinal for the command, however the remaining parameters are manufacturer specific.

*End of informative comment.*

**IDL Definition**

```
TCPA_RESULT TPM_FieldUpgrade(
      [in, out] TCPA_AUTH* ownerAuth,
      ...);
```

**Type**

TCPA protected capability; the TPM Owner must authenticate the command. This is an optional command and a TPM is not required to implement this command in any form.

**Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_AUTH | ownerAuth | Authentication from TPM owner to execute command |
| ... | | Remaining parameters are manufacturer specific |

**Actions**

The TPM SHALL perform the following when executing the command:

1. Validate the TPM Owners authorization to execute the command
2. Validate that the upgrade information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
3. Validate that the upgrade target is the appropriate TPM model and version.
4. Process the upgrade information and update the protected capabilities
5. Set the TCPA_PERSISTENT_DATA.revMajor and TCPA_PERSISTENT_DATA.revMinor to the values indicated in the upgrade. The selection of the value is a manufacturer option. The values MUST be monotonically increasing. Installing an upgrade with a major and minor revision that is less than currently installed in the TPM is a valid operation.
6. Set the TCPA_VOLATILE_FLAGS.deactivated to TRUE.

**Descriptions**

The upgrade mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of field upgrade. The TPM MUST NOT use the endorsement key for identification or encryption in the upgrade process. The upgrade process MAY use a TPM Identity to deliver upgrade information to specific TPM's.

The upgrade process can only change protected capabilities.

The upgrade process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TCPA specification, protection profiles and security targets after the upgrade. The upgrade MAY NOT decrease the security values from the original security target.

The security target used to evaluate this TPM MUST include this command in the TOE.

## 8.17 TPM_SetRedirection

*Informative comment:*

Redirected keys enable the output of a TPM to be directed to non-TCPA security functions in the platform without exposing that output to non-security functions.

It is sometimes desirable to direct the TPM's output directly to specific platform functions without exposing that output to other platform functions. To enable this, the key in a leaf node of TCPA Protected Storage can be tagged as a "redirect" key. Any plaintext output data secured by a redirected key is passed by the TPM directly to specific platform functions and is not interpreted by the TPM.

Since redirection can only affect leaf keys, redirection applies to: TPM_Unbind, TPM_Unseal, TPM_Quote, TPM_Sign.

*End of informative comments.*

**Type**

TCPA protected capability; the TPM MAY implement this command. The user MUST supply authorization to use the key pointed to by keyHandle.

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SetRedirection |
| 4 | 4 | | | TCPA_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can implement redirection. |
| 5 | 4 | 2s | 4 | UINT32 | C1 | Manufacturer parameter |
| 6 | 4 | 3s | 4 | UINT32 | C2 | Manufacturer parameter |
| 7 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TCPA_AUTHDATA | privAuth | The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetRedirection |

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Action**

1. The TPM SHALL validate the authorization to use the key pointed to by keyHandle.
2. The TPM SHALL verify that the key pointed to by keyHandle has the redirection flag set to TRUE. If FALSE the TPM SHALL return TCPA_FAIL.
3. The TPM SHALL set the key handle redirection parameters according to the values in parameters c1 and c2.
4. A key that is tagged as a "redirect" key MUST be a leaf key in the TCPA Protected Storage blob hierarchy. A key that is tagged as a "redirect" key CAN NEVER be a parent key.
5. Ouput data that is the result of a cryptographic operation using the private portion of a "redirect" key:
   a. MUST be passed to an alternate output channel
   b. MUST NOT be passed to the normal output channel
   c. MUST NOT be interpreted by the TPM.
6. The authorization response returns to the caller.

## 8.18 Key and Session Management

*Start of informative comment:*

To alleviate limited temporary key storage within a TPM, a key and its related context information can be cached outside the TPM. The cached key will be exported from the TPM inside a key context blob that is opaque data outside the TPM.

For the protection of the key context blob either a symmetric or an asymmetric cryptographic algorithm can be used. It is the responsibility of the TPM to assure the confidentiality and integrity of a key context blob.

Other key management commands can be implemented, but cannot touch data in TCPA shielded locations.

*End of informative comment.*

### 8.18.1 TPM_SaveKeyContext

*Start of informative comment:*

SaveKeyContext saves a loaded key outside the TPM. After creation of the key context blob the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.

*End of informative comment.*

**Type**

TCPA optional function; TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SaveKeyContext |
| 4 | 4 | | | TCPA_KEY_HANDLE | keyHandle | The key which will be kept outside the TPM |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | UINT32 | keyContextSize | The actual size of the outgoing key context blob. If the command fails the value will be 0 |
| 5 | <> | | | BYTE[] | keyContextBlob | The key context blob. |

**Description**

This command allows saving a loaded key outside the TPM. After creation of the KeyContextBlob, the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.

A TCPA protected capability belonging to the TPM that created a key context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a cryptographic technique MUST be generated using the TPM's random number generator. Any symmetric key MUST be used within the power-on session during which it was created, only.

A key context blob SHALL enable verification of the integrity of the contents of the blob by a TCPA protected capability.

A key context blob SHALL enable verification of the session validity of the contents of the blob by a TCPA protected capability. The method SHALL ensure that all key context blobs are rendered invalid if power to the TPM is interrupted.

### 8.18.2 TPM_LoadKeyContext

Start of informative comment:

LoadKeyContext loads a key context blob into the TPM previously retrieved by a SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.

End of informative comment.

**Type**

TCPA optional function; TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_LoadKeyContext |
| 4 | 4 | | | UINT32 | keyContextSize | The size of the following key context blob. |
| 5 | <> | | | BYTE[] | keyContextBlob | The key context blob. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | TCPA_KEY_HANDLE | keyHandle | The handle assigned to the key after it has been successfully loaded. |

**Description**

This command allows loading a key context blob into the TPM previously retrieved by a TPM_SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.

The contents of a key context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.

The contents of a key context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.

## 8.19 Authorization Context Management

*Start of informative comment:*

To alleviate limited temporary authorization session storage within a TPM, an authorization handle and its related context information can be cached outside the TPM. The cached authorization context will be exported from the TPM inside an authorization context blob that is opaque data outside the TPM.

For the protection of the authorization context blob either a symmetric or an asymmetric cryptographic algorithm can be used. It is the responsibility of the TPM to assure the confidentiality and integrity of a key context blob.

Other Authorization context commands can be implemented, but cannot touch data in TCPA shielded locations.

*End of informative comment.*

## 8.19.1 TPM_SaveAuthContext

*Start of informative comment:*

SaveAuthContext saves a loaded authorization session outside the TPM. After creation of the authorization context blob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

*End of informative comment.*

### Type

TCPA optional function; TCPA protected capability.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SaveAuthContext |
| 4 | 4 | | | TCPA_AUTHHANDLE | authandle | Authorization session which will be kept outside the TPM |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | UINT32 | authContextSize | The actual size of the outgoing authorization context blob. If the command fails the value will be 0. |
| 5 | <> | | | BYTE[] | authContextBlob | The authorization context blob. |

### Description

This command allows saving a loaded authorization session outside the TPM. After creation of the authContextBlob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

A TCPA protected capability belonging to the TPM that created an authorization context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a cryptographic technique MUST be generated using the TPM's random number generator. Any symmetric key MUST be used within the power-on session during which it was created, only.

An authorization context blob SHALL enable verification of the integrity of the contents of the blob by a TCPA protected capability.

An authorization context blob SHALL enable verification of the session validity of the contents of the blob by a TCPA protected capability. The method SHALL ensure that all authorization context blobs are rendered invalid if power to the TPM is interrupted.

### 8.19.2 TPM_LoadAuthContext

*Start of informative comment:*

LoadAuthContext loads an authorization context blob into the TPM previously retrieved by a SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

*End of informative comment.*

**Type**

TCPA optional function; TCPA protected capability.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_LoadAuthContext |
| 4 | 4 | | | UINT32 | authContextSize | The size of the following authorization context blob. |
| 5 | <> | | | BYTE[] | authContextBlob | The authorization context blob. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | TCPA_KEY_HANDLE | authHandle | The handle assigned to the authorization session after it has been successfully loaded. |

**Description**

This command allows loading an authorization context blob into the TPM previously retrieved by a TPM_SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

The contents of an authorization context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.

The contents of an authorization context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.

# 9. Subsystem Credentials

## 9.1 Introduction

*Start of informative comment:*

This section defines the credentials by which various entities vouch for a Trusted Platform, plus the Subsystem capabilities that are used during the creation of those credentials.

*End of informative comment.*

All credentials MUST use the TCPA_VERSION structure.

## 9.2 Endorsement

*Start of informative comment:*

A TPM only has one asymmetric endorsement key pair. Due to the nature of this key pair, both the public and private parts of the key have privacy and security concerns.

Exporting the PRIVEK from the TPM must not occur. This is for security reasons. The PRIVEK is a decryption key and never performs any signature operations.

Exporting the public PUBEK from the TPM under controlled circumstances is allowable. Access to the PUBEK must be restricted to entities that have a "need to know." This is for privacy reasons.

The PUBEK is tagged with TCPA_VERSION to indicate the version of the capability that created the key at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

Repeated access to the PUBEK of a TPM is desirable in the process of manufacturing TPMs and platforms. Unfortunately, repeated access to the PUBEK is a security concern (because the PUBEK is used to acquire ownership of the TPM) and may be a privacy concern.

The first call to TPM_CreateEndorsementKeyPair generates the endorsement key pair. After a successful completion of TPM_CreateEndorsementKeyPair all subsequent calls return TCPA_FAIL.

The TPM_ReadPubek returns the PUBEK only while the readPubek flag is TRUE. The owner can set the readPubek flag with an owner authorized command. In order to increase confidence that the PUBEK returned is in response to the command a simple challenge/response is built into the call to TPM_ReadPubek. The command returns a hash of a submitted nonce and the PUBEK.

*End of informative comment.*

The PRIVEK and PUBEK MUST be accessed only by protected capabilities whose definition explicitly requires access to those keys.

The PRIVEK and PUBEK MAY be created by a process other than the use of TPM_CreateEndorsementKeyPair. If so, the process MUST result in a TPM and endorsement key whose properties are the same as those of a genuine TPM and an endorsement key created by execution of TPM_CreateEndorsementKeyPair in that TPM.

- The process MUST result in the same TPM state as that created by execution of TPM_CreateEndorsementKeyPair.
- The process MUST guarantee correct generation, cryptographic strength, uniqueness, privacy, and installation into a genuine TPM, of the endorsement key.
- The TPME, when creating the Endorsement Certificate, MUST be satisfied that the described endorsement key does exist in a genuine TPM and was installed by a process that met or exceeded the assurances provided by a genuine TPM performing TPM_CreateEndorsementKeyPair.
- The process MUST be defined in the TOE of the security target in use to evaluate the TPM

### 9.2.1 TPM_CreateEndorsementKeyPair

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | *4* | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateEndorsementKeyPair |
| 4 | *20* | | | TCPA_NONCE | antiReplay | Arbitrary data |
| 5 | <> | | | TCPA_KEY_PARMS | keyInfo | Information about key to be created, this includes all algorithm parameters |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | <> | | | TCPA_PUBKEY | pubEndorsementKey | The public endorsement key |
| 5 | *20* | | | TCPA_DIGEST | checksum | Hash of pubEndorsementKey and antiReplay |

**Description**

| Type | Name | Description |
|---|---|---|
| TCPA_STORE_ASYMKEY | PRIVEK | This SHALL be the private key of the endorsement key pair. |
| TCPA_PUBKEY | PUBEK | This SHALL be the public key of the endorsement key pair. |

The PRIVEK SHALL exist only in a TCPA-shielded location.

If the data structure TPM_ENDORSEMENT_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

**Actions**

The first valid TPM_CreateEndorsementKeyPair command received by a TPM SHALL

1. Validate the keyInfo parameters for the key description
    a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For interoperability the key length SHOULD be 2048

b. If the algorithm type is other than RSA the strength provided by the key MUST be comparable to RSA 2048

c. The other parameters of keyInfo (signatureScheme etc.) are ignored.

2. Create a key pair called the "endorsement key pair" using a TCPA-protected capability. The type and size of key are that indicated by keyInfo
3. Create checksum by performing SHA1 on the concatenation of (PUBEK || antiReplay)
4. Store the PRIVEK.
5. Export the data structures PUBEK and checksum
6. Set TCPA_PERSISTENT_FLAGS -> CEKPUsed to TRUE

Subsequent calls to TPM_CreateEndorsementKeyPair SHALL return code TCPA_FAIL.

### 9.2.2 TPM_ReadPubek

**Type**

TCPA protected capability

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReadPubek |
| 4 | 20 | | | TCPA_NONCE | antiReplay | Arbitrary data |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | <> | | | TCPA_PUBKEY | pubEndorsementKey | The public endorsement key |
| 5 | 20 | | | TCPA_DIGEST | checksum | Hash of pubEndorsementKey and antiReplay |

**Description**

This command returns the PUBEK.

**Actions**

The TPM_ReadPubek command SHALL

1. If TCPA_PERSISTENT_FLAGS -> readPubek is FALSE return TCPA_DISABLED_CMD.
2. If no EK is present the TPM MUST return TCPA_NO_ENDORSEMENT
3. Create checksum by performing SHA1 on the concatenation of (PUBEK || antiReplay).
4. Export the PUBEK and checksum.

### 9.2.3 TPM_DisablePubekRead

Start of informative comment:

The TPM Owner may wish to prevent any entity from reading the PUBEK. This command sets the non-volatile flag so that the TPM_ReadPubek command always returns TCPA_DISABLED_CMD.

End of informative comment.

**Type**

TCPA protected capability; the user must present authorization from the TPM Owner.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisablePubekRead |
| 4 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisablePubekRead |
| 4 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters HMAC key: ownerAuth. |

**Actions**

This capability sets the TCPA_PERSISTENTFLAGS -> readPubek flag to FALSE.

### 9.2.4 TPM_OwnerReadPubek

**Type**

TCPA protected capability; caller must supply authorization from the TPM Owner

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerReadPubek |
| 4 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerReadPubek |
| 4 | <> | 3s | <> | TCPA_PUBKEY | pubEndorsementKey | The public endorsement key |
| 5 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Description**

This command returns the PUBEK.

**Actions**

The TPM_ReadPubek command SHALL

1. Validate the TPM Owner authorization to execute this command
2. Export the PUBEK

## 9.3 Generating a Trusted Platform Module Identity

*Start of informative comment*

The purpose of TPM_MakeIdentity is to create

- an asymmetric key pair within the Trusted Platform Module and
- evidence that the key pair is bound to a label

Only the Owner of the TPM has the privilege of creating a TPM identity. (An identity is not activated until the reception of the command TPM_ActivateIdentity.)

TPM_MakeIdentity communicates new authorization data to the TPM using almost the same process as Protected Storage uses to communicate new authorization data for blobs. Both processes require the creation of a TPM_OSAP session and the use of the sessions shared secret to XOR the new authorization data. The requirement for TPM_MakeIdentity is that the TPM_OSAP session must start with the TPM Owner authorization.

The authorization data will provide the ability to associate authorization sessions with the new identity in the future. The protection of the authorization data comes from the XOR having a one-time pad nature to it. If an attacker can determine the shared secret of the TPM_OSAP session then the attacker can learn the new value of the authorization data. For the case of identities, the owner is always the SRK, which in many cases has well-known authorization data. This would allow an attacker to determine what the shared secret was and hence what the value of the new authorization data is.

To avoid the problem with the SRK, the TPM_MakeIdentity command requires the TPM_OSAP session to use the TPM Owner as the authorization to establish the session. This creates a shared secret that only the TPM Owner and the TPM know and allows the proper protections when using the XOR for encryption.

A tpm_signature_Key must be known only to the TPM.

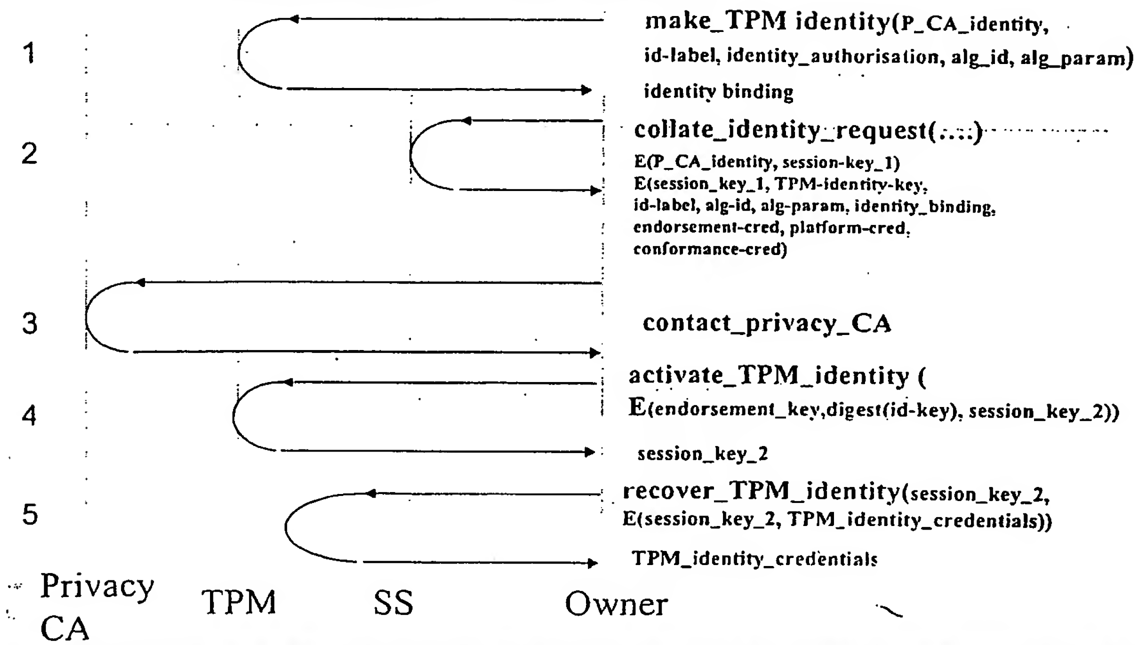
Identity_binding uses the private (signature) key of a TPM identity. The private (signature) key of a TPM identity is available only to selected commands. Its use enables a recipient to be certain that identity_binding was generated inside a TPM. This feature prevents a rogue Owner from assembling identity_binding data structures outside the TPM and hence obtaining attestation to the same TPM identity from multiple Privacy CAs.

Identity_binding is tagged with TCPA_VERSION so as to indicate the version of the capability that created the identity_binding at the time that identity_binding was generated. This may be useful in the event that capabilities are field-upgraded.

The algorithm parameter indicates the type of encryption algorithm in use for the TPM identity. It may indicate RSA, or ECC, to give two examples. The algorithm parameter indicates the parameters that are necessary for the particular encryption algorithm in use. For RSA, these parameters are just the length of the RSA key.

The PKI identity protocol enables a Trusted Platform Module to have multiple identities. Each identity may have attestation from exactly one Privacy CA.

# Obtaining a TPM identity




The TPM creates an identity-binding signature (the value of a signature over the TCPA_IDENTITY_CONTENTS structure). Among other things, this proves possession of the new private key, which does the signing of the TCPA_IDENTITY_CONTENTS structure. The Subsystem sends the signature along with evidence of a genuine TPM and the platform the TPM resides on to a Privacy CA. The encryption of the request is to provide privacy not security.

The Privacy CA inspects the evidence and concurs that the TPM is genuine and in a valid platform. The Privacy CA validates the signature of the TCPA_IDENTITY_CONTENTS structure and verifies that it was signed using the private key corresponding to the public key in the identity request. The TCPA_IDENTITY_CONTENTS structure includes a hash of the Privacy CA's public key. The Privacy CA obtains assurance that it (and not some other Privacy CA) is the target of the request to provide the identity attestation.

The Privacy CA cannot check that the public key inside identity-binding signature belongs to a genuine TPM, but it knows that the TPM described in the evidence is a genuine TPM. The Privacy CA generates the attestation credential and encrypts the credential for decryption by the requesting TPM. The Privacy CA also sends the genuine TPM a "statement" that the credential attests to a particular public key (the one in the identity-credential).

The TPM receives the encrypted data. It cannot parse the credential, but it can check that the credential attests to one of its public keys, by checking the "statement" from the Privacy CA. Only if the credential relates to one of the TPM's public keys does the TPM enable recovery of the credential.

The presumption is that the Privacy CA is trustworthy. This must be the case for the acceptance of the attestation by a third party. Hence, if the attestation is worth having, the "statement" from the Privacy CA to the TPM can be trusted. Hence, the TPM "knows" that the encrypted credential relates to the public key in the "statement". The Privacy CA has ensured that only a genuine TPM can recover the encrypted credential and statement, and that a genuine TPM will enable recovery of the credential only if the credential is associated with a public key belonging to the TPM.

A rogue can certainly pose as a Privacy CA and cause the TPM to release the credential created by that rogue. But who will trust the attestation provided by that rogue? A trustworthy credential can be recovered only if it attests to a public key of a genuine TPM, because the Privacy CA that created the credential can

be trusted to check that a TPM is genuine and to correctly state that a credential describes a particular public key, and a genuine TPM checks that the public key belongs to that TPM before releasing the credential.

The reason for including the hash of the public key of the Privacy CA inside identity-binding signature is to prevent a rogue obtaining attestation from multiple Privacy CAs. The identity-binding signature creation is an atomic operation performed at the same time as the key pair creation, and therefore the TPM cannot be coerced into creating a version of the identity-binding signature with the same keys but a different Privacy CA public key.

The identity-binding signature is one of the few operations that are permitted to use the private (signature) key of a TPM identity. A version of identity binding with a different Privacy CA public key can't be reproduced by commands from outside the TPM, because the TPM will refuse to sign arbitrary data with a private (signature) key of a TPM identity.

The process deliberately has certain characteristics:

For example, during TPM_MakeIdentity:

- The atomic generation of the key pair and encrypted identity binding information prevents the creation by a TPM of duplicate identity binding information while avoiding the need for a TPM to retain state.
- Signing with the private (signature) key of a TPM identity prevents the creation of duplicate "identity binding" information outside a TPM.
- When a Privacy CA receives data, it can use the data describing the new TPM identity to check that the request for attestation (if it came from a genuine TPM) is a unique request, use the endorsement credentials to check that a stated TPM is a genuine TPM, and use the platform credentials and conformance credentials to check that a stated platform is a genuine Trusted Platform. The Privacy CA *cannot*, however, verify that the new TPM identity was actually generated by that genuine TPM. On the assumption, however, that the new TPM identity was actually generated by a genuine TPM, the Privacy CA generates TPM_IDENTITY_CREDENTIALs and a statement that expresses a binding between that TPM_IDENTITY_CREDENTIAL and the new TPM identity. The Privacy CA then encrypts this information so that it can be recovered only by the genuine TPM described by the endorsement credentials.
- During TPM_ActivateIdentity, the genuine TPM checks that the encrypted TPM_IDENTITY_CREDENTIAL is bound to one of the TPM's identities and enables decryption of TPM_IDENTITY_CREDENTIAL *only* if that association exists. This last stage is critical but subtle, since the TPM has insufficient computing power to parse TPM_IDENTITY_CREDENTIAL and relies on the "statement" from the Privacy CA that a TPM_IDENTITY_CREDENTIAL is associated with a given identity.
- The entire process depends critically on the trustworthiness of the Privacy CA. If the Privacy CA is trustworthy, a plaintext TPM_IDENTITY_CREDENTIAL recovered by a TPM describes an identity of a genuine TPM. Otherwise, a TPM_IDENTITY_CREDENTIAL cannot be trusted. The Privacy CA must be trusted to make TPM_IDENTITY_CREDENTIAL only if the request for attestation is a unique request and the stated TPM and platform are genuine. The Privacy CA must be trusted never to reveal a plaintext copy of TPM_IDENTITY_CREDENTIAL and to be truthful when stating that a particular TPM_IDENTITY_CREDENTIAL is associated with a particular identity.

***End of informative comment***

### 9.3.1 TPM_MakeIdentity

**Type**

TCPA protected capability; user must provide authorizations from the TPM Owner and the SRK.

**Incoming Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_MakeIdentity. |
| 4 | *20* | 2s | *20* | TCPA_ENCAUTH | identityAuth | Encrypted usage authorization data for the new identity |
| 5 | *20* | 3s | *20* | TCPA_CHOSENID_HASH | labelPrivCADigest | The digest of the identity label and privacy CA chosen for the new TPM identity. (See 10.4.6 for details) |
| 6 | <> | 4s | <> | TCPA_KEY | idKeyParams | Structure containing all parameters of new identity key. pubKey.keyLength & idKeyParams.encData are both 0 |
| 7 | *4* | | | TCPA_AUTHHANDLE | srkAuthHandle | The authorization handle used for SRK authorization. |
| | | 2 H1 | *20* | TCPA_NONCE | srkLastNonceEven | Even nonce previously generated by TPM |
| 8 | *20* | 3 H1 | *20* | TCPA_NONCE | srknonceOdd | Nonce generated by system associated with srkAuthHandle |
| 9 | *1* | 4 H1 | *1* | BOOL | continueSrkSession | Ignored |
| 10 | *20* | | | TCPA_AUTHDATA | srkAuth | The authorization digest for the inputs and the SRK. HMAC key: srk.usageAuth. |
| 11 | *4* | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. Session type MUST be OSAP. |
| | | 2 H2 | *20* | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 12 | *20* | 3 H2 | *20* | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 13 | *1* | 4 H2 | *1* | BOOL | continueAuthSession | Ignored |
| 14 | *20* | | *20* | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| *PARAM* | | *HMAC* | | *Type* | *Name* | *Description* |
|---|---|---|---|---|---|---|
| *#* | *SZ* | *#* | *SZ* | | | |
| 1 | *2* | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | *4* | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | *4* | 1s | *4* | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | *4* | TCPA_COMMAND_CODE | ordinal | Command ordinal:TPM_ORD_MakeIdentity. |
| 4 | <> | 3s | <> | TCPA_KEY | idKey | The newly created identity key |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | 4 | 4s | 4 | UINT32 | identityBindingSize | The used size of the output area for identityBinding |
| 6 | <> | 5s | <> | BYTE[ ] | identityBinding | Signature of TCPA_IDENTITY_CONTENTS using idKey.private. |
| 7 | 20 | 2 H2 | 20 | TCPA_NONCE | srkNonceEven | Even nonce newly generated by TPM. |
| | | 3 H2 | 20 | TCPA_NONCE | srknonceOdd | Nonce generated by system associated with srkAuthHandle |
| 8 | 1 | 4 H2 | 1 | BOOL | continueSrkSession | Fixed value FALSE |
| 9 | 20 | | | TCPA_AUTHDATA | srkAuth | The authorization digest used for the outputs and srkAuth session. HMAC key: srk.usageAuth. |
| 10 | 20 | 2 H1 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 11 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Fixed value FALSE |
| 12 | 20 | | 20 | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Description**

The command TPM_MakeIdentity is used to generate an identity in a TPM and to request attestation to that identity.

The public key of the new TPM identity SHALL be identityPubKey. The private key of the new TPM identity SHALL be tpm_signature_key.

This command requires XOR encryption of the authorization to use the new identity. To create an XOR string, the caller takes the OSAP session shared secret, concatenates it with authLastNonceEven, and then hashes the result. This hash encrypts the authorization value and produces identityAuth.

**Properties of the new identity**

| Type | Name | Description |
|---|---|---|
| TCPA_PUBKEY | `identityPubKey` | This SHALL be the public key of a previously unused asymmetric key pair. |
| TCPA_STORE_ASYMKEY | `tpm_signature_key` | This SHALL be the private key that forms a pair with identityPubKey and SHALL be extant only in a TCPA-shielded location. |

This capability also generates a TCPA_KEY containing the tpm_signature_key.

If identityPubKey is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

**Actions**

A Trusted Platform Module that receives a valid TPM_MakeIdentity command SHALL do the following:

1. Validate the idKeyParams parameters for the key description
    a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For interoperability the key length SHOULD be 2048
    b. If the algorithm type is other than RSA the strength provided by the key MUST be comparable to RSA 2048

2. Use authHandle to verify that the Owner authorized all TPM_MakeIdentity input parameters.
3. Use srkAuthHandle to verify that the SRK owner authorized all TPM_MakeIdentity input parameters.
4. Verify that idKeyParams -> keyUsage is TPM_KEY_IDENTITY. If it is not, return TCPA_BAD_PARAMETER"
5. Verify that idKeyParams -> keyFlags -> migratable is FALSE. If it is not, return TCPA_BAD_PARAMETER"
6. Obtain the identity_authorization to be associated with the new TPM identity, by decrypting the field IdentityAuth. The establishment of the TPM_OSAP session MUST use the authentication of the TPM Owner.
7. Set continueAuthSession to FALSE.
8. Create an asymmetric key pair (identityPubKey and tpm_signature_key) using a TCPA-protected capability, in accordance with the algorithm specified in idKeyParams
9. Create TCPA_KEY structure idKey using idKeyParams as the default values for the structure.
10. Ensure that the authorization information in identityAuth is properly stored in the idKey as usageAuth.
11. Attach identityPubKey and tpm_signature_key to idKey
12. Set idKey -> migrationAuth to TTCPA_PERSISTANT_DATA -> tpmProof
13. Ensure that all TCPA_PAYLOAD_TYPE structures identity this key as TCPA_PT_ASYM
14. Encrypt the private portion of idKey using the SRK as the parent key
15. Create a TCPA_IDENTITY_CONTENTS structure named idContents using labelPrivCADigest and the information from idKey
16. Sign idContents using tpm_signature_key and TCPA_SS_RSASSAPKCS1v15_SHA1. Store the result in identityBinding.

### 9.3.2 TSS_CollateIdentityRequest

*Start of informative comment:*

The purpose of the TSS_CollateIdentityRequest command is to assemble all the data that will be required by a Privacy CA in order to assess a platform and attest to the identity of a Subsystem.

The TSS_CollateIdentityRequest command is separate from the TPM_MakeIdentity command because their processing might be done on different engines. The reason is that TSS_CollateIdentityRequest does not have to be trustworthy but TPM_MakeIdentity must be trustworthy. Therefore, an implementation of TSS_CollateIdentityRequest does not require the same protection as an implementation of TPM_MakeIdentity.

A session key (a nonce) is used to provide confidentiality of the "TCPA_IDENTITY_REQ". This is to ensure that only the Privacy CA chosen by the Owner can interpret the data, while minimizing exposure of that Privacy CA's identity (public) key.

Once the data structure TCPA_IDENTITY_REQ has been produced, it should be sent to the Privacy CA chosen by the Owner.

*End of informative comment.*

**Type**

TSS capability and MAY be TPM capability.

**Suggested Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_IDENTITY_PROOF | `proof` | This SHALL be the structure specified in 4.30.3 |
| TCPA_KEY_PARMS | `SymAlgorithm` | This SHALL specify the type of symmetric encryption algorithm to be used for a session key, and the scheme it will use to perform encryptions. |
| TCPA_PUBKEY | `CaPubKey` | This SHALL be public key of the CA which will provide the credential for the identity |
| UINT32* | `ReqSize` | This SHALL be the size of the identityReq field |
| TCPA_IDENTITY_REQ* | `IdentityRequest` | This SHALL be the data structure defined in this section. |

**Description**

The command TSS_CollateIdentityRequest assembles all data necessary to request attestation of a Trusted Platform Module identity.

The structure "proof" (of type TPM_IDENTITY_PROOF) contains fields that a privacy-CA requires in order to decide whether to attest to the TPM identity described by "proof".

A Trusted Platform Subsystem that receives a valid TSS_CollateIdentityRequest command SHALL export the data structure "TCPA_IDENTITY_REQ."

The TSS in executing this function performs two encryptions. The first is to symmetrically encrypt the information and the second is to encrypt the symmetric encryption key with an asymmetric algorithm. The symmetric key is a random nonce and the asymmetric key is the public key of the CA that will provide the identity credential.

For reasons of interoperability, CaPubKey SHOULD indicate TCPA_ALG_RSA (RSA) with a key length of 2048 bits. SymAlgorithm SHOULD be TCPA_ALG_3DES (3DES in CBC mode).

The use of TCPA_ALG_AES (AES in CBC mode) as the symmetric algorithm is encouraged.

**Actions**

The command SHALL perform the following actions:

1. Validate that the TSS can support the symmetric algorithm and the asymmetric algorithm necessary to perform the encryptions. If the TSS does not support these algorithms it MUST return TCPA_BAD_PARAMETER.
2. Initialize the identityRequest area to be the TCPA_IDENTITY_REQ structure.
3. Create a session key in accordance with the algorithm in SymAlgorithm, by calling TSS_GetRandom.
4. Create an IV in accordance with the algorithm in SymAlgorithm, by calling TSS_GetRandom.
5. Encrypt the TCPA_IDENTITY_PROOF structure using the session key created in step 3, the IV created in step 4, and the symmetric algorithm specified by SymAlgorithm.
6. Place the encrypted TCPA_IDENTITY_PROOF blob into the TCPA_IDENTITY_REQ.symBlob field.
7. Create a TCPA_SYMMETRIC_KEY structure using the session key created in step 3.
8. Encrypt the TCPA_SYMMETRIC_KEY structure created in step 7 using the algorithm specified in the key caPubKey.
9. Place the encrypted TCPA_SYMMETRIC_KEY blob into the TCPA_IDENTITY_REQ.asymBlob field.
10. Create TCPA_IDENTITY_REQ.SymAlgorithm using SymAlgorithm and inserting the IV created in step 4 into the previously empty "parms" field.
11. Create TCPA_IDENTITY_REQ.AsymAlgorithm from CaPubKey.
12. Return the TCPA_IDENTITY_REQ structure.

### 9.3.3 Contacting a Privacy CA

***Start of informative comment***

The operations and procedures of a Privacy CA are outside the scope of this specification.

The anticipation, however, is that a Privacy CA will use at least the following checks before agreeing to attest to a TPM identity for a platform:

- Interpret the data structure "TCPA_IDENTITY_REQ" in the supplied data and validate the various fields in the structure.
- The verification of the privacy CA's public is inherent in the decryption of the TCPA_IDENTITY_REQ structure. If the decryptions yield valid structures then the key was correct otherwise the structures are not properly formed and the key was bad.
- Interpret the conformance credential information in the supplied data in order to verify that the design of the platform meets the TCPA specification and is in accordance with the policies of the Privacy CA.
- Interpret the platform credential information in the supplied data in order to verify that the construction of the platform meets the TCPA specification and is in accordance with the policies of the Privacy CA.
- Interpret the endorsement credential information in the supplied data in order to verify that the construction of the TPM meets the TCPA specification and is in accordance with the policies of the Privacy CA.
- Create a TCPA_IDENTITY_CONTENTS structure and validate the signature of the area provided by the new identity.

It is anticipated that a Privacy CA will then take the following actions:

1. Using the supplied data, construct a TPM identity credential according to the TCPA specification, and sign the instantiation using a private key belonging to the Privacy CA.
2. Generate a session key. The assumption is that the session key comes from a suitable random number generator that provides a suitable level of entropy.
3. Create the TCPA_SYM_CA_ATTESTATION structure.
4. Store the session key in TCPA_ASYM_CA_CONTENTS.
5. Create a digest of the identityPubKey. Store the digest value in TCPA_ASYM_CA_CONTENTS.
6. Encrypt the TCPA_ASYM_CA_CONTENTS structure using the PUBEK sent in the attestation request.
7. Return the TCPA_SYM_CA_ATTESTATION structure and the encrypted TCPA_ASYM_CA_CONTENTS structure.

The symmetric algorithm should be the same algorithm that the TSS used in creating the TCPA_IDENTITY_REQ structure. The asymmetric algorithm must be the algorithm that is defined by the type of PUBEK.

***End of informative comment***

### 9.3.4 TPM_ActivateIdentity

*Start of informative comment:*

The purpose of TPM_ActivateIdentity is to twofold. The first purpose is to obtain assurance that the credential in the TCPA_SYM_CA_ATTESTATION is for this TPM. The second purpose is to obtain the session key used to encrypt the TPM_IDENTITY_CREDENTIAL.

TPM_ActivateIdentity checks that the symmetric session key corresponds to a TPM-identity before releasing that session key.

Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is required to authorize the TPM_ActivateIdentity command. The owner may authorize the command using either the TPM_OIAP or TPM_OSAP authorization protocols.

*End of informative comment.*

**Type**

TCPA protected capability; user must provide authorization from the TPM Owner to execute command.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ActivateIdentity. |
| 4 | 4 | | | TCPA_KEY_HANDLE | idKey | Identity key to be activated |
| 5 | 4 | 2s | 4 | UINT32 | blobSize | Size of encrypted blob from CA |
| 6 | <> | 3s | <> | BYTE [ ] | blob | The encrypted ASYM_CA_CONTENTS structure |
| 7 | 4 | | | TCPA_AUTHHANDLE | idKeyAuthHandle | The authorization handle used for ID key authorization. |
| | | 2 H1 | 20 | TCPA_NONCE | idKeyLastNonceEven | Even nonce previously generated by TPM |
| 8 | 20 | 3 H1 | 20 | TCPA_NONCE | idKeynonceOdd | Nonce generated by system associated with idKeyAuthHandle |
| 9 | 1 | 4 H1 | 1 | BOOL | continueIdKeySession | Continue usage flag for idKeyAuthHandle. |
| 10 | 20 | | | TCPA_AUTHDATA | idKeyAuth | The authorization digest for the inputs and ID key. HMAC key: idKey.usageAuth |
| 11 | 4 | | | TCPA_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2 H2 | 20 | TCPA_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 12 | 20 | 3 H2 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 13 | 1 | 4 H2 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 14 | 20 | | 20 | TCPA_AUTHDATA | ownerAuth | The authorization digest for inputs and owner. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | HMAC | Type | Name | Description |
|---|---|---|---|---|

| # | SZ | # | SZ | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1s | 4 | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2s | 4 | TCPA_COMMAND_CODE | ordinal | Command ordinal:TPM_ORD_ActivateIdentity. |
| 4 | <> | 3s | <> | TCPA_SYMMETRIC_KEY | symmetricKey | The decrypted symmetric key. |
| 5 | 20 | 2H1 | 20 | TCPA_NONCE | idKeyNonceEven | Even nonce newly generated by TPM. |
| | | 3H1 | 20 | TCPA_NONCE | idKeynonceOdd | Nonce generated by system associated with idKeyAuthHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueIdKeySession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TCPA_AUTHDATA | idKeyAuth | The authorization digest used for the returned parameters and idKeyAuth session. HMAC key: idKey.usageAuth. |
| 8 | 20 | 2H2 | 20 | TCPA_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H2 | 20 | TCPA_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H2 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 10 | 20 | | 20 | TCPA_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Description**

The command TPM_ActivateIdentity activates a TPM identity created using the command TPM_MakeIdentity.

The command assumes the availability of the private key associated with the identity. The command will verify the association between the keys during the process.

The command will decrypt the TCPA_ASYM_CA_CONTENTS structure, extract the session key and verify the connection between the public and private keys.

**Actions**

A Trusted Platform Module that receives a valid TPM_ActivateIdentity command SHALL do the following:

1. Using the authHandle field, validate the owner's authorization to execute the command and all of the incoming parameters.
2. Using the idKeyAuthHandle, validate the authorization to execute command and all of the incoming parameters
3. Decrypt blob using PRIVEK as the decryption key. The resulting decrypted area MUST be a TCPA_ASYM_CA_CONTENTS structure.
4. Compute a digest of the public key in the idKey. Compare the computed digest to the value in the decrypted TCPA_ASYM_CA_CONTENTS structure. Return with the error code TCPA_BAD_PARAMETER on a mismatch.
5. Validate that the idKey is the public key of a valid TPM identity by checking that idKey -> keyUsage is TPM_KEY_IDENTITY
6. Return the session key from the TCPA_ASYM_CA_CONTENTS structure.

### 9.3.5 TSS_RecoverTPMIdentity

*Start of informative comment:*

The purpose of TSS_RecoverIdentity is to recover a plaintext copy of the data structure TPM_IDENTITY_CREDENTIAL that attests that a particular identity belongs to a genuine TCPA Trusted Platform.

The TSS_RecoverIdentity command is separate from the TPM_ActivateIdentity command because their processing might be done on different engines. The reason is that TSS_RecoverIdentity does not have to be trustworthy, but TPM_ActivateIdentity must be trustworthy. Therefore, an implementation of TSS_RecoverIdentity does not require the same protection as an implementation of TPM_ActivateIdentity.

Exactly one entity may attest to a TPM identity.

Access to the TPM_IDENTITY_CREDENTIAL must be restricted to entities that have a "need to know". This is for reasons of privacy.

*End of informative comment.*

The command TSS_RecoverIdentity obtains a plaintext copy of the TPM_IDENTITY_CREDENTIAL created by a Privacy CA.

If the data structure TPM_IDENTITY_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is only available to authorized entities.

**Suggested Parameters**

| Type | Name | Description |
|---|---|---|
| TCPA_SYMMETRIC_KEY | SessionKey | This SHALL be the symmetric key decrypted by the TPM_ActivateIdentity |
| UINT32 | symAttSize | This SHALL be the size of the symAtt parameter |
| TCPA_SYM_CA_ATTESTATION* | symAtt | This SHALL be the TCPA_SYM_CA_ATTESTATION structure |
| UINT32* | CredentialSize | This SHALL be the size of the credential |
| BYTE* | Credential | This SHALL be the decrypted TCPA_IDENTITY_CREDENTIAL |

**Actions**

A Trusted Platform Subsystem that receives a valid TSS_RecoverIdentity command SHALL do the following:

1. Using the session key and the symmetric algorithm indicated by algorithm and the algorithm parameters, decrypt credential parameter inside TCPA_SYM_CA_ATTESTATION to recover the TPM_IDENTITY_CREDENTIAL.
2. The TSS SHOULD verify the self-consistency of TPM_IDENTITY_CREDENTIAL and abandon this TSS_RecoverIdentity process if there is an inconsistency. The process of verifying certificates is outside the scope of this specification.
3. Export TPM_IDENTITY_CREDENTIAL.

## 9.4 Instantiation of Data When Contacting a Privacy CA

*Start of informative comment:*

Unambiguous definition of data structures is necessary if those data are to be communicated between platforms. An ASN.1 description is such an unambiguous definition.

This section describes the protocol messages to be sent from the Owner to the Privacy CA and from the Privacy CA to the Owner during the procedure for obtaining a TPM identity. These messages will need to be supported by suitable transport (and lower-layer) protocols. A number of alternatives exist for the transport layer, including TCP, HTTP, e-mail, and FTP. However, specification of any of these alternatives — including resolution of related issues such as naming/addressing, whether polling should be allowed, and whether confirmation messages are required — is considered beyond the scope of this document.

Some of the data that is passed from the Privacy CA to the Owner is DER-encoded and must be used by the TPM. This is not, however, a significant burden for the TPM.

The Owner receives from the Privacy CA the ASN.1 DER-encoded structure PCAResponse, which is a SEQUENCE of version, symmAlg, encTcpaAsymCaContents, and tcpaSymCaAttestation. The Owner software (perhaps the TSS, or perhaps some other module) parses this structure, pulls out the encTcpaAsymCaContents field (which is a (tag, length, value) combination), and returns the value portion (which is simply a string of bits).

The Owner passes this "value" to the TPM. This "value", as stated in the specification, is the ciphertext resulting from the encryption, under the PUBEK, of a DER-encoded structure. Therefore, the TPM simply decrypts the value it is handed using its PRIVEK. The resulting string of bits has the following format:

- tag1 length1 tag2 length2 value2 tag3 length3 value3
- The first field ("tag1") is an identifier for SEQUENCE and takes up one byte. The next field ("length1") reports the number of octets (i.e., bytes) remaining in the entire string, and also takes up one byte. "tag2" is an identifier for BIT STRING and takes up one byte. "length2" reports the length in bytes of "value2" and takes up two bytes. "value2" is the result of hashing tpmIdKey (e.g., if SHA-1 is used, it is 160 bits in length, but the TPM will already know this so it doesn't need to understand "length2" in order to figure this out). "tag3" is an identifier for BIT STRING and takes up one byte. "length3" reports the length in bytes of "value3" and takes up two bytes. "value3" is the symmetric key. (Note that "value3" may have a length of 128 bits for one symmetric cipher, 168 for another, and 256 for yet another, but the TPM does not need to determine this from "length3". Instead, it simply reads to the end of the string).

In short, therefore, the TPM does the following on decryption:

- skips five bytes
- reads the next (say, 160, if SHA-1 is used) bits and compares this to the table of hashed, inactivated public keys that it has stored(if there is a match it proceeds, otherwise, it aborts the operation)
- skips the next three bytes
- reads the remaining bytes (until the end of the string) into a buffer, and
- returns this buffer to the Owner as the symmetric key.

*End of informative comment.*

### 9.4.1 From Owner to Privacy CA

The protocol from the Owner to the Privacy CA SHALL consist of the following IdentityRequest message:

```
TcpaIdentityReq ::= SEQUENCE {
      version           Version,
      asymAlg           TcpaAlgorithmParms,
      symAlg            TcpaAlgorithmParms,
```

```
      asymBlob          EncTcpaSymmetricKey,
      symBlob           EncTcpaIdentityProof
}
Version ::= INTEGER
-- the version number, for compatibility with future revisions of
-- this specification. It shall be 0 for this version of the
-- specification.

TcpaAlgorithmParms ::= SEQUENCE {
      algId             AlgorithmIdentifier,
      parms             OCTET STRING
      -- the parameters for the algorithm specified in algId
}

EncTcpaSymmetricKey ::= BIT STRING
-- the ciphertext resulting from the encryption (under the public
-- identity key of the Privacy CA) of the following DER-encoded data
-- structure.

TcpaSymmetricKey ::= SEQUENCE {
      algId             AlgorithmIdentifier,
      encScheme         OCTET STRING, -- TCPA_ENCRYPTION_SCHEME
      data              BIT STRING  -- randomly-generated session key
}

EncTcpaIdentityProof ::= BIT STRING
-- the ciphertext resulting from the encryption (under the session
-- key in TcpaSymmetricKey above) of the following DER-encoded data
-- structure:

TcpaIdentityProof ::= SEQUENCE {
      tcpaVersion       TCPASpecVersion,   -- "major.minor"
      tpmIdKey          SubjectPublicKeyInfo,   -- new public key
      tpmIdLabel        OCTET STRING,      -- identity label
      identityBinding   BIT STRING,        -- (see below)
      endorsementCred   Certificate,       -- X.509v3 PK cert
      platformCred      Certificate,       -- X.509 attr. cert
      conformanceCred   Certificate        -- X.509 attr. cert
}

-- SubjectPublicKeyInfo
-- (a SEQUENCE of an AlgorithmIdentifier and a BIT STRING) is
-- specified in X.509.  The BIT STRING contains the subject's public
-- key (for example, if the algorithm specified is rsaEncryption, the
-- BIT STRING contains the BER encoding of a value of PKCS #1 type
-- "RSAPublicKey").

-- identityBinding
-- is the signature value(using the newly generated TPM private key
-- that corresponds to the public key in tpmIdKey) over the data
-- specified in Section 4.30.1 TCPA_IDENTITY_CONTENTS. How that data -- is
formatted or delimited is beyond the scope of the protocol
-- specified  here; however, the formatting chosen must be known to
-- both the TPM and the Privacy CA.
```

### 9.4.2 From Privacy CA to Owner

The protocol from the Privacy CA to the Owner consists of the PCAResponse message:

```
PCAResponse ::= SEQUENCE {
      version                  Version,
      symmAlg                  AlgorithmIdentifier,
      encTcpaAsymCaContents    EncTcpaAsymCaContents,
      tcpaSymCaAttestation     TcpaSymCaAttestation
}

EncTcpaAsymCaContents ::= BIT STRING
-- the ciphertext resulting from the encryption (under the PUBEK of
-- the TPM) of the following DER-encoded data structure:

TcpaAsymCaContents ::= SEQUENCE {
      idDigest                 BIT STRING, -- hash of tpmIdKey
      sessionKey               BIT STRING
}

-- NOTE: the validity of the entire protocol for obtaining a TPM
-- identity depends critically upon the assumption that a  genuine
-- TPM will only ever decrypt data using its PRIVEK as part of the
-- TPM_ActivateIdentity() call.  An Owner will never be able to ask a
-- TPM for the decryption  of arbitrary data that has been encrypted
-- with its PUBEK. Furthermore, the difficulty of successfully
-- impersonating a TPM is ultimately bound to the computational
-- complexity of finding a collision for idDigest. It is therefore
-- STRONGLY RECOMMENDED that the digest be computed  using the full
-- output of a cryptographic hash algorithm of  sufficient strength
-- (e.g., the full 160 bits of SHA-1).


TcpaSymCaAttestation ::= SEQUENCE {
      algorithm                TcpaAlgorithmParms,
      encCredential            BIT STRING
      -- the ciphertext resulting from the encryption (under the
      -- symmetric session key in TcpaAsymCaContents above) of the
      -- tpmIdentityCredential (which is itself DER-encoded as an
      -- X.509 PK Certificate).
}
```

## 9.5 Instantiation of Credentials as Certificates

*Start of informative comment:*

Unambiguous definition of a data structure containing credentials is necessary if those credentials are to be communicated between platforms. A certificate is such an unambiguous definition.

The TCPA requires credentials to prove various pieces of information. This version of the specification uses X.509 certificates to provide these credentials. The TCPA is not requiring the entire flexibility of X.509; rather, TCPA is using the well defined certificate structure to create the necessary TCPA credentials.

*End of informative comment.*

**Certificate syntax**

TCPA certificate syntax conforms with the definitions for public-key certificates and attribute certificates in X.509. The following TCPA certificate types are public-key certificates:

- TPM endorsement certificate
- TPM identity certificate

The following TCPA certificate types are attribute certificates:

- Platform endorsement certificate
- Platform conformance certificate
- Validation data certificate

The form of the following certificates is out of scope for this version of the TPM specification:

- TPM endorsement entity certificate
- TCPA component endorsement entity certificate
- Platform endorsement entity certificate
- Platform conformance certificate

The serial number used by the following certificates is not unique for each platform. It is anticipated that the serial number would remain the same on multiple platforms.

For instance, all platforms of the same model and version would have the same serial number in their platform endorsement credential. For these same platforms, the platform conformance certificates would all use the same serial number but that number would be different than the endorsement certificate serial number.

### 9.5.1 Instantiation of TPM_ENDORSEMENT_CREDENTIALs

*Start of informative comment:*

An endorsement certificate is an instantiation of an TPM_ENDORSEMENT_CREDENTIAL.

Access to an endorsement certificate must be restricted to entities that have a "need to know." This is for reasons of privacy.

This definition assumes that the PUBEK is a 2048bit RSA keys.

*End of informative comment.*

If the data structure <endorsement_certificate> is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

**Overview**

The TPM endorsement certificate represents an assertion by the TPM endorsement entity that the referenced TPM conforms with the TCPA TPM specification.

**Profile**

Notes:

- Some fields are assigned a value even though the certificate user performs no action based on that value. In such cases, the intention is to inhibit non-TCPA implementations from making inappropriate use of the certificate.
- It is intended that the lifetime of a TPM will be shorter than the crypto-period of the TPM endorsement public and private keys. Therefore, keys are not "rolled-over".
- The trustworthiness of the architecture is vulnerable to the compromise of a single TPM endorsement private key. However, the architecture does not include a revocation mechanism. Nevertheless, certain forms of revocation scheme can be retrofitted, should it become necessary at some time in the future.

In the case of the TPM endorsement certificate, the ***issuer*** is the TPM endorsement entity and the ***user*** is a Privacy CA.

| Field | Issuer action | User action |
|---|---|---|
| Version | Assign value 2 (v3). | Check value = 2, else reject. |
| Serial number | Assign a value unique amongst all certificates issued by "issuer". | Use in validating the platform endorsement and conformance certificates. |
| Signature | Assign the algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5). | Check the algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the TPME (which shall be a 2048-bit RSA key), obtained by an out-of-band means and referenced by "issuer" and "authority key identifier". |
| Issuer | The distinguished name of the TPM endorsement entity. That is the entity that asserts that the subject TPM conforms with the TCPA specification. (Note: this may be the TPM manufacturer or a conformance test laboratory.) | Check that the name is the name of one of the acceptable TPM endorsement entities, use in validating the platform endorsement and conformance certificates. |

| | | |
|---|---|---|
| Validity | Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme). | Check that the current time is later than the notBefore time, else reject. |
| Subject | Assign the value NULL. | No action. |
| Subject public key info | Assign algorithm identifier RSAES-OAEP (1:2:840:113549:1:1:7). Include a 2048-bit RSA public key for key encipherment with OAEP formatting. (Note: this is the TPM public endorsement key.) | Use the public key in the TPM identity protocol. |
| Issuer unique identifier | Omit. | No action. |
| Subject unique identifier | Omit. | No action. |
| Extensions | | |
| Authority key identifier | Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the manufacturer's certificate, if available, else omit. | Use to locate the certificate that contains a public key of the manufacturer with which the signature on this certificate can be verified. |
| Subject key identifier | Omit. | No action. |
| Key usage | May be omitted. If included, then the key encipherment bit shall be set TRUE. | If present, then check that the key encipherment bit is TRUE, else reject. |
| Extended key usage | Omit. | If present and marked critical, then reject. |
| Private key usage period | Omit. | If present, then check that the current time is later than the notBefore time. |
| Certificate policies | Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Assign the cPSuri policy qualifier the value of an HTTP URL at which a plain language version of the TPM endorsement entity's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Trusted Platform Module Endorsement". | Check that at least one acceptable policyIdentifier value is present. Transfer the acceptable policyInformation value to the TPM identity certificate "certificate policies" extension. |
| Policy mappings | Omit. | No action. |
| Subject alternative name | Assign "critical" the value FALSE. Include the TPM identity, using the directory name-form with RDNs for the TPM manufacturer, model and version numbers. | Check that the TPM manufacturer, model and version numbers are acceptable. Transfer to the TPM identify certificate "subject alternative name" extension value for the TPM. |
| Issuer alternative name | Omit. | No action. |

| | | |
|---|---|---|
| Subject directory attributes | Include a "subject directory attributes" extension. Assign "critical" the value FALSE. Include the multi-valued attribute "supported algorithms" (see X.509). Include object identifiers for the following algorithms: RSAES-OAEP, SHA-1 (1.3.14.3.2.26) and TPM identity protocol. | Adapt the TPM identity protocol to use only algorithms supported by the TPM. |
| | Include the "TCPA Specification Version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the TPM implementation conforms. | Check that the TCPA specification version is acceptable, else reject. |
| | Optionally, include the "security qualities" attribute with a text string reflecting the security qualities of the TPM. (Note: this is the TPM distributed validation.) | Optionally (and if present), check whether the TPM implementation has acceptable security qualities. Transfer to the TPM identity certificate "subject directory attributes" extension. |
| Basic constraints | Assign "critical" the value TRUE. Assign "CA" the value FALSE | No action. |
| Name constraints | Omit. | No action. |
| Policy constraints | Omit. | No action. |
| Inhibit any policy | Omit. | No action. |
| CRL distribution points | Omit. | If present and marked critical, then reject. |

### 9.5.2 Instantiation of PLATFORM_CREDENTIAL

*Start of informative comment:*

A platform certificate is an instantiation of a platform credential.

Access to the platform certificate must be restricted to entities that have a "need to know". This is for reasons of privacy.

*End of informative comment.*

If the data structure <platform_certificate> is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

**Overview**

The Platform Endorsement Certificate represents an assertion by the platform endorsement entity that the referenced platform incorporates a TPM and an RTM in a manner that conforms with the TCPA specification.

**Profile**

Note: some fields are assigned a value even though the certificate user performs no action with that value. In such cases, the intention is to inhibit non-TCPA implementations from making inappropriate use of the certificate.

In the case of the Platform endorsement certificate, the *issuer* is the platform manufacturer and the *user* is a Privacy CA.

| Field | Issuer action | User action |
|---|---|---|
| Version | Assign value 1 (v2). | Check value = 1, else reject. |
| Holder | BaseCertificateID referencing the corresponding TPM endorsement certificate. (Note: this is the TPM credential reference.) | Check that the certificate ID correctly references the TPM endorsement certificate used to validate the TPM identity request message, else reject. |
| Issuer | The distinguished name of the platform endorsement entity. That is the entity that asserts that the subject platform incorporates a TPM and RTM in a manner that conforms with the TCPA specification. (Note: this may be the platform manufacturer or a conformance test laboratory.) | Check that the name is the name of one of the acceptable platform endorsement entities. |
| Signature | Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5). | Check algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the Platform Endorsement Entity (which should be a 2048-bit RSA key), obtained by an out-of-band means and referenced by "issuer" and "authority key identifier" |
| Serial number | Assign a value unique per instance of a TBB amongst all certificates issued by "issuer" | No action. |
| attrCertValidity Period | Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the | Check that the current time is later than the notBefore time. else reject. |

| | | |
|---|---|---|
| | the latest time permitted by the encoding scheme). | |
| Attributes | A "supported algorithms" attribute (see X.509) indicating the cryptographic algorithms supported by the platform. | Transfer the object identifiers for any acceptable algorithms to the TPM identity certificate "subject directory attributes" extension. |
| | Include the "TCPA Specification Version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the platform implementation conforms. | Check that the TCPA specification version is acceptable, else reject. |
| | If the TPM has been successfully evaluated against a Common Criteria protection profile, then include the TPM protection profile identifier attribute. | Optionally, check whether the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate. |
| | If the TPM has been successfully evaluated against a Common Criteria security target, then include the TPM security target identifier attribute. | Optionally, check whether the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate. |
| | If the RTM and the means by which the TPM and RTM have been incorporated into the platform have been successfully evaluated against a Common Criteria protection profile, then include the "foundation protection profile" identifier attribute. | Optionally, check whether the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate "subject directory attributes" extension. |
| | If the RTM and the means by which the TPM and RTM have been incorporated into the platform have been successfully evaluated against a Common Criteria security target, then include the "foundation security target" identifier attribute. | Optionally, check whether the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate "subject directory attributes" extension. |
| | If there is, or will be, a Platform Conformance Certificate, then a ConformanceCertificateLocation attribute should be included to indicate how, and from where, it can be retrieved. | Use the information to locate and retrieve the corresponding Platform Conformance Certificate. |
| | Optionally, include the "security qualities" attribute with a text string reflecting the security qualities of the platform. (Note: this is the platform distributed validation.) | Optionally (and if present), check whether the platform implementation has acceptable security qualities. Transfer to the TPM identity certificate "subject directory attributes" extension. |
| Issuer unique identifier | Omit. | No action. |
| Extensions | | |
| Certificate | Assign "critical" the value TRUE. | Check that at least one acceptable |

| | | |
|---|---|---|
| policies | Assign policyIdentifier at least one object identifier. Assign the cPSuri policy qualifier the value of an HTTP URL at which a plain language version of the platform manufacturer's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Trusted Platform Endorsement". | policyIdentifier value is present. Transfer the policyInformation value to the TPM identity certificate "certificate policies" extension. |
| Subject alternative name | Assign "critical" the value FALSE. Include the platform name, uniquely identifying the type of the platform with RDNs for the manufacturer, model and version numbers. | Check that the manufacturer, model and version numbers are acceptable. Transfer to the TPM identity certificate "subject alternative name" extension. |
| Authority key identifier | Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the platform endorsement entity certificate, if available, else omit. | The certificate user may use this value to locate the certificate that contains a public key of the platform endorsement entity with which the signature on this certificate can be verified. |
| SOA Identifier | Omit. | No action. |
| Authority Attribute Identifier | Omit. | No action. |
| Role Specification Certificate Identifier | Omit. | No action. |
| Basic Attribute Constraints | Assign "critical" the value TRUE. Assign "authority" the value FALSE. | Check that "authority" is FALSE. |
| Delegated Name Constraints | Omit. | No action. |
| Time Specification | Omit. | No action. |
| Acceptable Certificate Policies | Assign "critical" the value TRUE. Assign one or more of the values of policyIdentifier from the certificate policies extension of the TPM endorsement certificate. | Check that the certificate policies extension of the TPM endorsement certificate contains at least one of the values. |
| Attribute Descriptor | Omit. | No action. |
| User Notice | Omit. | No action. |
| No Rev Available | Omit. | No action. |
| Acceptable Privilege Policies | Omit. | No action. |

### 9.5.3 Instantiation of TPM_CONFORMANCE_CREDENTIAL

**Overview**

The Platform Conformance Certificate represents an assertion by the platform conformance entity that the referenced platform conforms with the TCPA specification.

**Profile**

Note: some fields are assigned a value even though the certificate user performs no action with that value. In such cases, the intention is to inhibit non-TCPA implementations from making inappropriate use of the certificate.

In the case of the Platform conformance certificate, the ***issuer*** is the platform manufacturer and the ***user*** is a Privacy CA.

| Field | Issuer action | User action |
|---|---|---|
| Version | Assign value 1 (v2). | Check value = 1, else reject. |
| Holder | Include the platform name, uniquely identifying the type of the platform with RDNs for the manufacturer, model and version numbers. | Check that the value is the same as the value in the corresponding Platform Endorsement Certificate, Subject Alternative Name extension, else reject. |
| Issuer | The distinguished name of the platform conformance entity. That is the entity that asserts that the design of the platform conforms with the TCPA specification. (Note: this may be the platform manufacturer or a conformance test laboratory.) | Check that the name is the name of one of the acceptable platform conformance entities. |
| Signature | Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5). | Check algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the platform conformance entity (which should be a 2048-bit RSA key), obtained by an out-of-band means and referenced by "issuer" and "authority key identifier". |
| Serial number | Assign a value unique per evaluated series of a TBB amongst all certificates issued by "issuer" | No action. |
| attrCertValidity Period | Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme). | Check that the current time is later than the notBefore time, else reject. |
| Attributes | Include a "supported algorithms" attribute (see X.509) indicating the algorithms supported by the platform. | Transfer the object identifiers for any acceptable algorithms to the TPM identity certificate "subject directory attributes" extension. |
| | Include the "TCPA specification version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the platform implementation | Check that the TCPA specification version is acceptable, else reject. |

| | | |
|---|---|---|
| | conforms. | |
| | If the TPM has been successfully evaluated against a Common Criteria protection profile, then include the TPM protection profile identifier attribute. | Check that the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate. |
| | If the TPM has been successfully evaluated against a Common Criteria security target, then include the TPM security target identifier attribute. | Check that the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate. |
| | If the RTM and means by which the RTM and TPM are incorporated into the platform has been successfully evaluated against a Common Criteria protection profile, then include the foundation protection profile identifier attribute. | Check that the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate "subject directory attributes" extension. |
| | If the RTM and the means by which the RTM and TPM have been incorporated into the platform have been successfully evaluated against a Common Criteria security target, then include the foundation security target identifier attribute. | Check that the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate "subject directory attributes" extension. |
| Issuer unique identifier | Omit. | No action. |
| Extensions | | |
| Certificate policies | Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Assign the cPSuri policy qualifier the value of an HTTP URL at which a plain language version of the platform conformance entity's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Conformance Credential". | Check that at least one acceptable policyIdentifier value is present. Transfer the policyInformation value to the TPM identity certificate. |
| Subject alternative name | Assign "critical" the value FALSE. Include the platform name, uniquely identifying the type of the platform with RDNs for the platform manufacturer, model and version numbers. | Check that the manufacturer, model and version numbers are identical to those in the platform endorsement certificate "subject alternative name" extension. |
| Authority key identifier | Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the platform conformance entity's public-key certificate, if available, else omit. | The certificate user may use this value to locate the certificate that contains a public key of the platform conformance entity with which the signature on this certificate can be verified. |
| SOA Identifier | Omit. | No action. |

| | | |
|---|---|---|
| Authority Attribute Identifier | Omit. | No action. |
| Role Specification Certificate Identifier | Omit. | No action. |
| Basic Attribute Constraints | Assign "critical" the value TRUE. Assign "authority" the value FALSE. | Check that "authority" is FALSE. |
| Delegated Name Constraints | Omit. | No action. |
| Time Specification | Omit. | No action. |
| Acceptable Certificate Policies | Omit. | No action. |
| Attribute Descriptor | Omit. | No action. |
| User Notice | Omit. | No action. |
| No Rev Available | Omit. | No action. |
| Acceptable Privilege Policies | Omit. | No action. |

### 9.5.4 Instantiation of VALIDATION_DATA

Start of informative comment

A Validation Data Attribute Certificate is an instantiation of validation data

End of informative comment

**Overview**

The validation data certificate represents an assertion by the component validation entity that the component instructions referenced by the certificate have the attributes conveyed in the certificate. The certificate syntax conforms with the X.509 definition for an attribute certificate.

In the case of the validation certificate, the ***issuer*** is the Validation Entity and the ***user*** is a TPS.

| Field | Issuer action | User action |
|---|---|---|
| Version | Assign value 1 (v2). | Check value = 1, else reject. |
| Holder | ObjectDigestInfo with missing object identifier. The value of objectDigest shall be the digest calculated over the memory image of the software instructions using the identified digest algorithm. | Calculate the digest of the memory image of the software instructions and check that it is identical to the value in this field prior to passing control to the component, else reject. |
| Issuer | The distinguished name of the component validation entity. That is the entity that asserts that the component exhibits the attributes contained in the certificate. (Note: typically, but not necessarily, the manufacturer of the component). | Check that the name is the name of one of the acceptable component validation entities. |
| Signature | Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5). | Check algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the software manufacturer (which should be a 2048-bit RSA key), obtained by an out-of-band means and referenced by "issuer" and "authority key identifier". |
| Serial number | Assign a value unique amongst all certificates issued by "issuer". Uniqueness to be determined by the manufacturer. | No action. |
| attrCertValidityPeriod | Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme). | Check that the current time is later than the notBefore time, else reject. |
| Attributes | Include the "TCPA specification version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the component conforms. | Check that the TCPA specification version is acceptable, else reject. |
| | Optionally, include the "security qualities" attribute with a text string reflecting the security qualities of the component. (Note: this is the component distributed | Optionally (and if present), check whether the component implementation has acceptable security qualities. |

| | | |
|---|---|---|
| | validation.) | |
| Issuer unique identifier | Omit. | No action. |
| Extensions | | |
| Certificate policies | Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Assign the cPSuri policy qualifier the value of an HTTP URL at which a plain language version of the component conformance entity's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Validation Data". | Check that at least one acceptable policyIdentifier value is present. |
| Subject Alternative Name | Assign "critical" the value FALSE. Include the component name, using the "component name" attribute, with RDNs for the component manufacturer, model and version numbers. | May be used to determine whether or not the component is trustworthy. |
| Authority key identifier | Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the component validation entity certificate, if available, else omit. | The certificate user may use this value to locate the certificate that contains a public key of the component validation entity with which the signature on this certificate can be verified. |
| SOA Identifier | Omit. | No action. |
| Authority Attribute Identifier | Omit. | No action. |
| Role Specification Certificate Identifier | Omit. | No action. |
| Basic Attribute Constraints | Assign "critical" the value TRUE. Assign "authority" the value FALSE. | Check that "authority" is FALSE. |
| Delegated Name Constraints | Omit. | No action. |
| Time Specification | Omit. | No action. |
| Acceptable Certificate Policies | Omit. | No action. |
| Attribute Descriptor | Omit. | No action. |
| User Notice | Omit. | No action. |
| No Rev Available | Omit. | No action. |
| Acceptable Privilege Policies | Omit. | No action. |

| | | |
|---|---|---|
| Privilege Policies | | |

### 9.5.5 Instantiation of TPM_IDENTITY_CREDENTIAL

*Start of informative comment:*

A TPM identity certificate is an instantiation of a TPM_IDENTITY_CREDENTIAL.

Access to the TPM identity certificate must be restricted to entities that have a "need to know." This is for reasons of privacy.

This definition assumes that TPM identity keys are 2048bit RSA keys.

*End of informative comment.*

If the data structure <TPM identity certificate> is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

**Overview**

The TPM identity certificate represents an assertion by the Privacy CA that the referenced TPM identity is controlled by a TPM that conforms with the TPM specification. It contains a different public key to that contained in the TPM endorsement certificate, but it contains identifying and policy information transferred from the TPM endorsement, platform endorsement and platform conformance certificates.

**Profile**

Note:

- Some fields are assigned a value even though the certificate user performs no action with that value. In such cases, the intention is to inhibit non-TCPA implementations from making inappropriate use of the certificate.
- The policies identified in the TPM and platform certificates are represented by oids and are not distinguishable except by reference to the contents of the policies themselves. The verifier, however, must be able to distinguish between the different policy types.

In the case of the TPM identity certificate, the ***issuer*** is the Privacy CA and the ***user*** is an integrity verifier.

| Field | Issuer action | User action |
|---|---|---|
| Version | Assign value 2 (v3). | Check value = 2, else reject. |
| Serial number | Assign a value unique amongst all certificates issued by "issuer". | No action. |
| Signature | Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5). | Check the algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the Privacy CA (which should be a 2048-bit RSA key), obtained by an out-of-band means and referenced by "issuer" and "authority key identifier". |
| Issuer | The distinguished name of the Privacy CA. | Check that the name is the name of an acceptable Privacy CA. |
| Validity | Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme). | Check that the current time is later than the notBefore time, else reject. |
| Subject | NULL. | No action. |
| Subject public | Assign algorithm identifier sha- | Check algorithm identifier = |

| | | |
|---|---|---|
| key info | 1WithRSAEncryption (1:2:840:113549:1:1:5). The 2048-bit RSA public key provided to the Privacy CA by the TPM owner in the identity request message. | 1:2:840:113549:1:1:5, else reject. Use the public key in the integrity verification procedure. |
| Issuer unique identifier | Omit. | No action. |
| Subject unique identifier | Omit. | No action. |
| Extensions | | |
| Authority key identifier | Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the Privacy CA's public-key certificate, if available, else omit. | The certificate user may use this value to locate the certificate that contains a public key of the Privacy CA with which the signature on this certificate can be verified. |
| Subject key identifier | Omit. | No action. |
| Key usage | May be omitted. If included, then the digital signature bit shall be set TRUE. | If present, then check that the digital signature bit is TRUE, else reject. |
| Extended key usage | Omit. | If present and marked critical, then reject. |
| Private key usage period | Omit. | If present, then check that the current time is later than the notBefore time, else reject. |
| Certificate policies | Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Optionally, assign the cPSuri the value of an HTTP URL at which a plain language version of the Privacy CA's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Trusted Platform Identity". Also, include the policyInformation values from the certificate policies extensions of the TPM endorsement and platform endorsement and conformance certificates provided in the TPM identity request message. | Check that at least one acceptable Privacy CA policyIdentifier value is present. Optionally, check that at least one acceptable TPM endorsement, one acceptable platform endorsement and one acceptable platform conformance policyIdentifier value are present. |
| Policy mappings | Omit. | No action. |
| Subject alternative name | Assign "critical" the value FALSE. Include three values in the extension:<br><br>The TPM manufacturer, model and version numbers from the TPM endorsement certificate "subject alternative name" extension provided in the TPM identity request message;<br><br>The platform manufacturer, model | Check that the manufacturer, model and version numbers of the TPM and of the platform are acceptable. |

| | | |
|---|---|---|
| | and version numbers from the platform endorsement certificate "subject alternative name" extension provided in the TPM identity request message; and<br><br>The TPM identity label provided to the Privacy CA by the TPM owner in the identity request message, encoded as a TPMIdLabel other-name. The TPM owner should choose a label syntax and semantics that are understood by the integrity verifier. (Note: the specified syntax accommodates multi-byte character sets). | |
| Issuer alternative name | Omit. | No action. |
| Subject directory attributes | Assign "critical" the value FALSE. Include a multi-valued "supported algorithms" (see X.509) attribute containing object identifiers from the "subject directory attributes" extension of the TPM endorsement certificate and the "attributes" field of the platform endorsement certificate and the platform conformance certificate provided in the TPM identity request message. | Adapt the integrity verification protocol to use only algorithms supported by the TPM and the associated platform. |
| | Include the single-valued "TPM protection profile" attribute from the platform endorsement certificate provided in the TPM identity request message. | Check that the identifier is acceptable. |
| | Include the single-valued "TPM security target" attribute from the platform endorsement certificate provided in the TPM identity request message. | Check that the identifier is acceptable. |
| | Include the single-valued "Foundation protection profile" attribute from the platform endorsement certificate provided in the TPM identity request message. | Check that the identifier is acceptable. |
| | Include the single-valued "Foundation security target" attribute from the platform endorsement certificate provided in the TPM identity request message. | Check that the identifier is acceptable. |
| | Include the "security qualities" attribute from the TPM endorsement certificate provided in the TPM identity request message. (Note: this is the | Optionally (and if present), check whether the TPM has acceptable security qualities. |

| | | |
|---|---|---|
| | TPM distributed validation.) | |
| | Include the "security qualities" attribute from the platform endorsement certificate provided in the TPM identity request message. (Note: this is the platform distributed validation.) | Optionally (and if present), check whether the platform has acceptable security qualities. |
| | Include the "tcpaVersion" attribute provided in the TPM identity request message. | Check that the TCPA specification version is acceptable, else reject. |
| Basic constraints | Assign "critical" the value TRUE. Assign "CA" the value FALSE. | No action. |
| Name constraints | Omit. | No action. |
| Policy constraints | Omit. | No action. |
| Inhibit any policy | Omit. | No action. |
| CRL distribution points | Omit. | If present and marked critical, then reject. |

### 9.5.6 ASN.1 Definitions

*Start of informative comment:*

The intention is to register TCPA as an international body in the ISO registration hierarchy. This will lead to shorter oids (object identifiers) and gives TCPA autonomy in the management of its own object identifiers.

*End of informative comment.*

The syntax of the "security qualities" attribute is as follows:

```
SecurityQualities ATTRIBUTE ::= {
      WITH SYNTAX SecurityQualities
      ID tcpa-tpmSecurityQualities }

SecurityQualities ::= SEQUENCE {
      version INTEGER, --0 for this version of the attribute syntax --
      statement [0]      UTF8String }
```

Note: future versions of this certificate profile may define additional, optional, "security qualities" fields. Inclusion of the "statement" field will remain mandatory.

The syntax of the "TCPA Specification Version" attribute is as follows:

```
TCPASpecVersion ATTRIBUTE ::= {
      WITH SYNTAX TCPASpecVersion
      ID tcpa-specVersion }

TCPASpecVersion ::= SEQUENCE {
      major INTEGER,
      minor INTEGER }
```

The syntax of the protection profile and security target attributes is as follows:

```
TPMProtectionProfile ATTRIBUTE ::= {
      WITH SYNTAX ProtectionProfile
      ID tcpa-at-tpmProtectionProfile }

TPMSecurityTarget ATTRIBUTE ::= {
      WITH SYNTAX SecurityTarget
      ID tcpa-at-tpmSecurityTarget }

FoundationProtectionProfile ATTRIBUTE ::= {
      WITH SYNTAX ProtectionProfile
      ID tcpa-at-foundationProtectionProfile }

FoundationSecurityTarget ATTRIBUTE ::= {
      WITH SYNTAX SecurityTarget
      ID tcpa-at-foundationSecurityTarget }
      ProtectionProfile ::= OBJECT IDENTIFIER
      SecurityTarget ::= OBJECT IDENTIFIER
```

The syntax of the "component name" attribute is as follows:

```
ComponentName ATTRIBUTE ::= {
      WITH SYNTAX Name
      ID tcpa-at-componentName }
```

The following definitions define the syntax of the RDNs used in the subject alternative name extension to identify the type of the TPM and the platform.

```
TpmManufacturer ATTRIBUTE ::= {
      WITH SYNTAX UTF8String
      ID tcpa-at-tpmManufacturer }

TpmModel ATTRIBUTE ::= {
      WITH SYNTAX UTF8String
      ID tcpa-at-tpmModel }

TpmVersion ATTRIBUTE ::= {
      WITH SYNTAX UTF8String
      ID tcpa-at-tpmVersion }

PlatformManufacturerl ATTRIBUTE ::= {
      WITH SYNTAX UTF8String
      ID tcpa-at-platformManufacturer }

PlatformModel ATTRIBUTE ::= {
      WITH SYNTAX UTF8String
      ID tcpa-at-platformModel }

PlatformVersion ATTRIBUTE ::= {
      WITH SYNTAX UTF8String
      ID tcpa-at-platformVersion }

TPMIdLabel OTHER-NAME ::= {UTF8String IDENTIFIED BY {tcpa-at-tpmIdLabel}}
```

--Object identifier assignments—

```
tcpa                                 OBJECT IDENTIFIER ::= {TBD}
tcpa-specVersion                     OBJECT IDENTIFIER ::= {tcpa-1}
tcpa-attribute                       OBJECT IDENTIFIER ::= {tcpa-2}
tcpa-protocol                        OBJECT IDENTIFIER ::= {tcpa-3}
tcpa-at-tpmManufacturer              OBJECT IDENTIFIER ::= {tcpa-attribute 1}
tcpa-at-tpmModel                     OBJECT IDENTIFIER ::= {tcpa-attribute 2}
tcpa-at-tpmVersion                   OBJECT IDENTIFIER ::= {tcpa-attribute 3}
tcpa-at-platformManufacturer         OBJECT IDENTIFIER ::= {tcpa-attribute 4}
tcpa-at-platformModel                OBJECT IDENTIFIER ::= {tcpa-attribute 5}
tcpa-at-platformVersion              OBJECT IDENTIFIER ::= {tcpa-attribute 6}
tcpa-at-componentManufacturer        OBJECT IDENTIFIER ::= {tcpa-attribute 7}
tcpa-at-componentModel               OBJECT IDENTIFIER ::= {tcpa-attribute 8}
tcpa-at-componentVersion             OBJECT IDENTIFIER ::= {tcpa-attribute 9}
tcpa-at-securityQualities            OBJECT IDENTIFIER ::= {tcpa-attribute 10}
tcpa-at-tpmProtectionProfile         OBJECT IDENTIFIER ::= {tcpa-attribute 11}
tcpa-at-tpmSecurityTarget            OBJECT IDENTIFIER ::= {tcpa-attribute 12}
tcpa-at-foundationProtectionProfile OBJECT IDENTIFIER ::= {tcpa-attribute 13}
tcpa-at-foundationSecurityTarget     OBJECT IDENTIFIER ::= {tcpa-attribute 14}
tcpa-at-tpmIdLabel                   OBJECT IDENTIFIER ::= {tcpa-attribute 15}
tcpa-prt-tpmIdProtocol               OBJECT IDENTIFIER ::= {tcpa-protocol 1}
```

# 10. Conformance Criteria

## 10.1 Base Levels for Interoperability

*Start of informative comment:*

The TCPA Support Services (TSS) will interoperate with other TSS devices and applications external to the TPM. The functions that interoperate are identity creation, challenge and response, backup, and maintenance. The interoperability must be at a level so that an application or other TSS can, without modification, send messages and receive replies. The messaging system may be either real-time or store-and-forward.

The use of TPM and TSS is intentional in the conformance section. The difference between the two is the level of protection that is available for the functions or data. The TPM provides tight control over execution and data access, but for the TSS there is no such requirement.

To achieve maximum flexibility, the TSS supports a negotiation protocol. This protocol allows the requestor to determine which features are available and the parameter settings that are appropriate for each of them.

There is no guarantee of interoperability when support for additional algorithms and protocols is provided.

*End of informative comment.*

The algorithms and protocols in this specification are the REQUIRED algorithms and protocols. A TPM subsystem MAY support additional algorithms and protocols. When this specification specifies the use of the TSS for a feature, an implementation MAY place the feature in the TPM.

The interoperability requirements shall be implemented at the TSS layer not the TPM. It is the responsibility of the TPM manufacturer to produce a vendor specific byte stream generator. The TSS will provide a generic API that all applications for a specific platform (PC, PDA, etc) can use.

## 10.2 Conformance Specification Sheet

*Start of informative comment:*

This section provides a quick listing of the protocols and algorithms that a TPM must support. For details review the section specific to the function in question.

Algorithms

- RSA, SHA-1, HMAC

Operations

- Random number generation
- Key generation
- Digital signatures (signing and verification)
- Protected storage
- Auditing
- Volatile memory
- Non-volatile memory

*End of informative comment.*

## 10.3 Protocol Negotiation and Algorithm Agility

*Start of informative comment:*

The TPM requires interoperability between devices when sending migration packets, identities and backup issues. For these reasons the specification mandates algorithms and message formats.

A related issue is that the set of algorithms picked by the specification may not meet the needs of a certain community. The specification therefore allows different algorithms to be in use. For instance, when creating an identity the creator can specify the algorithm and algorithm parameters for the identity. The specification requires that the TPM support the RSA algorithm, however the TPM may support additional algorithms and parameters.

Any challenger can request the list of algorithms and parameters that a TPM supports using the TPM_GetCapability command.

A challenger does not negotiate algorithms and parameters rather the challenger requests a specific type and the TPM either executes the command or fails the request.

*End of informative comment.*

The TPM MUST support the base algorithms specified for each operation. The TPM MAY support additional algorithms and parameters.

The TPM manufacturer MUST include in the TPM credential all algorithms that the TPM supports.

The TSS manufacturer MUST include in the platform credential all algorithms that the TSS supports.

## 10.4 Cryptographic Algorithms and Protocols

*Start of informative comment:*

The algorithms and protocols are the minimum that the TSS and TPM must support. Additional algorithms and protocols may be available to the TPM and TSS. All algorithms and protocols available in the TPM and TSS must be included in the list in the TPM and platform credential.

*End of informative comment.*

### 10.4.1 Asymmetric

*Start of informative comment:*

The asymmetric algorithm provides both digital signatures and wrapping of keys. The requirement of the TPM to support RSA allows the specification of one algorithm for both purposes.

TPM devices that implement different algorithms may have different algorithms perform the signing and wrapping.

There is no requirement concerning how the RSA algorithm is to be implemented. TPM manufacturers may use Chinese Remainder Theorem (CRT) implementations or any other method. Designers should review P1363 for guidance on RSA implementations.

*End of informative comment.*

- The TPM MUST support RSA.
- The TPM MUST use the RSA algorithm for encryption and digital signatures.
- The TPM MUST support key sizes of 512, 1024, and 2048 bits. The TPM MAY support other key sizes. The minimum RECOMMENDED key size is 1024 bits.
- The RSA public exponent MUST be e, where $e = 2^{16}+1$.

TPM devices that use CRT as the RSA implementation MUST provide protection and detection of failures during the CRT process to avoid attacks on the private key.

The TPM MAY implement other asymmetric algorithms such as DSA or elliptic curve. These algorithms may be in use for wrapping, signatures, and other operations. There is no guarantee that these keys can migrate to other TPM devices or that other TPM devices will accept signatures from these additional algorithms.

All Storage keys MUST be of strength equivalent to a 2048 bits RSA key or greater. The TPM SHALL NOT load a Storage key whose strength less than that of a 2048 bits RSA key.

All TPM Identity keys MUST be of strength equivalent to a 2048 bits RSA key, or greater.

### 10.4.2 Symmetric

*Start of informative comment:*

The encryption done by the TPM does not require a symmetric algorithm. The TSS must provide the bulk encryption support. The assumption is that the TSS has larger bandwidth and more MIPS to accomplish this type of encryption.

There is no requirement that a TPM NOT support a symmetric algorithm. A TPM may implement a symmetric algorithm.

The requirement to support both DES and 3DES is because some localities have restrictions on the import or export of 3DES and the TSS should not have an export or import limitation. DES should be in use only when the 3DES is not allowable.

*End of informative comment.*

The TSS MUST support 3DES. 3DES SHOULD be the symmetric algorithm of choice. The key size of 3DES MUST be 196 bits (three 64-bit keys). 3DES MUST be run in encrypt-decrypt-encrypt (EDE) mode. The TSS MUST provide detection of weak 3DES keys.

The TSS MUST support DES. The key size for DES MUST be 64 bits (56 bits plus parity). The TSS MUST provide detection of weak DES keys.

The TSS SHOULD have support for AES when it becomes available.

A TPM MUST support the storage of at least 256-bit symmetric keys.

### 10.4.3 Hashing

The TPM MUST support the SHA-1 hash algorithm as defined by FIPS-180-1. The output of SHA-1 is 160 bits and all areas that expect a hash value are REQUIRED to support the full 160 bits.

### 10.4.4 Signature Operations

The TPM MUST use the RSA algorithm for signature operations.

The TPM MAY use other asymmetric algorithms for signatures; however, there is no requirement that any other TPM device either accept or verify those signatures.

The TPM MUST use P1363 for the format and design of the signature output.

### 10.4.5 Creating a PCR composite hash

The definition specifies the operation necessary to create TCPA_COMPOSITE_HASH.

**Action**

The hashing MUST be done using the SHA-1 algorithm.

The input must be a valid TCPA_PCR_SELECTION structure.

The process creates a TCPA_PCR_COMPOSITE structure from the TCPA_PCR_SELECTION structure and the PCR values to be hashed. If constructed by the TPM the values MUST come from the current PCR registers indicated by the PCR indices in the TCPA_PCR_SELECTION structure.

The process then computes a SHA-1 digest of the TCPA_PCR_COMPOSITE structure.

The output is the SHA-1 digest just computed.

### 10.4.6 Creating TCPA_CHOSENID_HASH

This definition specifies the operation necessary to create a TCPA_CHOSENID_HASH structure.

**Parameters**

| Type | Name | Description |
|---|---|---|
| BYTE [] | `identityLabel` | The label chosen for a new TPM identity |
| TCPA_PUBKEY | `privacyCA` | The public key of a privacy CA chosen to attest to a new TPM identity |

**Action**

The hashing MUST be done using the SHA-1 algorithm.

The process concatenates identityLabel and privacyCA (identityLabel followed by privacyCA) and computes a SHA-1 digest of the concatenated data.

The output is the SHA-1 digest just computed.

### 10.4.7 Using Secret Keys

*Informative comments:*

Secret keys can be loaded into a TPM, but preferably are generated inside the TPM.

A TPM generated key must not be used as a secret key if it has already been exposed.

Secret keys obtained from blobs must not be exposed outside the TPM.

*End of informative comments.*

A secret key is a key that is a private asymmetric key or a symmetric key.

Data SHOULD NOT be used as a secret key by a TCPA protected capability unless that data has been extant only in a shielded location.

A key generated by a TCPA protected capability SHALL NOT be used as a secret key unless that key has been extant only in a shielded location.

A secret key obtained by a TCPA protected capability from a Protected Storage blob SHALL be extant only in a shielded location.

## 10.5 Random Number Generator (RNG)

*Start of informative comment:*

The Random Number Generator (RNG) is the source of randomness in the TPM. The TPM uses these random values for nonces, key generation and randomness in signatures.

The understanding is that this definition of the RNG, depending on implementation, could be a Pseudo Random Number Generator (PRNG). On those devices that have a hardware source of entropy, this implementation may be an RNG and not a PRNG so there is no need for to keep track of which is which, that is, the specification will always use RNG.

*End of informative comment.*

The RNG for the TPM will consist of the following components:

- Entropy source and collector
- State register
- Mixing function

The RNG capability is a TPM-protected capability with no access control.

The RNG output may or may not be shielded data. When the data is for internal use by the TPM (e.g., asymmetric key generation) the data MUST be held in a shielded location. When the data is for use by the TSS or another external caller, the data is not shielded.

### 10.5.1 Entropy Source and Collector

*Start of informative comment:*

The entropy source is the process or processes that provide entropy. These types of sources could include noise, clock variations, air movement, and other types of events.

The entropy collector is the process that collects the entropy, removes bias, and smoothes the output. The difference between the collector and the mixing function (described in section 10.6.3, "Mixing Function") is that the collector may have special code to handle any bias or skewing of the raw entropy data. For instance, if the entropy source has a bias of creating 60 percent 1s and only 40 percent 0s, then the collector design takes that bias into account before sending the information to the state register.

*End of informative comment.*

The entropy source MUST provide entropy to the state register in a manner that provides entropy that is not visible to an outside process. For compliance purposes, the entropy source MAY be in the TSS and not the TPM; however, attention MUST be paid to the reporting mechanism.

The entropy source MUST provide the information only to the state register. The entropy source may provide information that has a bias, so the entropy collector must remove the bias before updating the state register. The bias removal could use the mixing function or a function specifically designed to handle the bias of the entropy source. The entropy source can be a single device (such as hardware noise) or a combination of events (such as disk timings). It is the responsibility of the entropy collector to update the state register whenever the collector has additional entropy.

### 10.5.2 State Register

*Start of informative comment:*

The state register implementation may use two registers: a non-volatile register and a volatile register. The TPM loads the volatile register from the non-volatile register on startup. Each subsequent change to the state register from either the entropy source or the mixing function affects the volatile state register. The TPM saves the current value of the volatile state register to the non-volatile register on TPM power

down. The TPM may update the non-volatile register at any other time. The reasons for using two registers are:

- to handle an implementation in which the non-volatile register is in a flash device and
- to avoid overuse of the flash as the number of writes to a flash device are limited.

*End of informative comment.*

The state register is in a TPM-shielded location. The state register MUST be non-volatile. The update function to the state register is a TPM-protected capability. The primary input to the update function SHOULD be the entropy collector.

If the current value of the state register is unknown, calls made to the update function with known data MUST NOT result in the state register ending up in a state that an attacker could know. This requirement implies that the addition of known data MUST NOT result in a decrease in the entropy of the state register.

The TPM MUST NOT export the state register.

### 10.5.3 Mixing Function

*Start of informative comment:*

The mixing function takes the state register and produces some output.

The mixing function is a TPM-protected capability. The mixing function takes the state register and creates the output of the RNG. The output MUST conform to the requirements for PRNG from FIPS 140-1.

*End of informative comment.*

Each use of the mixing function MUST affect the state register. This requirement is to affect the volatile register and does not need to affect the non-volatile state register.

### 10.5.4 RNG Reset

*Start of informative comment:*

The resetting of the RNG occurs at least in response to a loss of power to the device.

These tests prove only that the RNG is still operating properly; they do not prove how much entropy is in the state register. This is why the self-test checks only after the load of previous state and may occur before the addition of more entropy.

*End of informative comment.*

The RNG MUST NOT output any bits after a system reset until the following occurs:

- The entropy collector performs an update on the state register. This does not include the adding of the previous state but requires at least one bit of entropy.
- The mixing function performs a self-test. This self-test MUST occur after the loading of the previous state. It MAY occur before the entropy collector performs the first update.

## 10.6 Key Generation

*Start of informative comment:*

Key generation is algorithm specific. The requirements for a given algorithm come from the preceding section or sections specific to it.

There are no timing requirements on the length of time that a TPM must meet when performing key generation.

*End of informative comment.*

### 10.6.1 Asymmetric

The TPM MUST generate asymmetric key pairs. The generate function is a protected capability and the private key is held in a shielded location. The implementation of the generate function MUST be in accordance with P1363.

The prime-number testing for the RSA algorithm MUST use the definitions of P1363. If additional asymmetric algorithms are available, they MUST use the definitions from P1363 for the underlying basis of the asymmetric key (for example, elliptic curve fitting).

### 10.6.2 Symmetric

The TSS MUST generate a symmetric key by taking the next *n* bits from the TPM RNG.

The TSS SHOULD provide any processing of a symmetric key. Processing is an algorithm-specific operation and implementation is left to the designer.

### 10.6.3 Nonce Creation

The creation of all nonce values MUST use the next *n* bits from the TPM RNG.

## 10.7 Auditing

*Start of informative comment:*

The TPM and TSS must be able to report a log of events. The log uses the same paradigm as the PCRs: the TPM keeps a PCR value that extends for each log event, and the TSS maintains the log entries for Challengers to review.

The TPM generates an audit event and the TSS creates the log. The protection of the log is a TSS requirement. The TSS is responsible for collecting each audit log event.

The TPM uses a PCR and extends it for each audit event. The TSS can use the PCR to create a log that shows any attempt to tamper with it.

The TPM Owner can select the operations that will generate an audit event.

*End of informative comment.*

The TPM MUST be able to generate audit events for all TCPA protected capabilities.

The TPM Owner MUST be able to select the functions that will generate an audit event at any time.

The TPM MUST provide a PCR to store and log the audit events. The TPM MUST allow for the reporting of the current audit log PCR value. The value that the TPM adds to the TPM audit PCR MUST be the TCPA_AUDIT_EVENT structure.

The TSS MUST provide a log of all TPM-generated events. The TPM will generate the event and the TSS will fill in the event details. The TPM SHALL provide as much detail as it has available; however, the TSS MUST fill in all remaining details for the audit event. For instance, the audit event will require a data and time stamp on the event. There is no requirement for a clock function in the TPM, so the date and time would come normally from the TSS.

The TPM MAY generate audit events for other functions and activities not on this list.

## 10.8 Self-Tests

The TPM MUST provide startup self-tests. The TPM MUST provide mechanisms to allow the self-tests to be run on demand. The response from the self-tests is pass or fail.

The TPM MUST complete the startup self-tests in a manner and timeliness that allows the TPM to be of use to the BIOS during the collection of integrity metrics. The TPM MUST complete the required checks before a given feature is in use. This requirement allows the TPM to test the integrity metric storage and allow its use while simultaneously continuing to test the signature engine.

There are two sections of startup self-tests: required and recommended. The recommended tests are not a requirement due to timing constraints. The TPM manufacturer should perform as many tests as possible in the time constraints.

The TPM MUST report the tests that it performs.

The TPM MUST provide a mechanism to allow self-test to execute on request by any Challenger.

The TPM MUST provide for testing of some operations during each execution of the operation.

### 10.8.1 Required Self-Tests

The TPM MUST check the following:

- RNG functionality. This test follows FIPS 140-1, which checks the functioning of an RNG.
- Reading and extending the integrity registers. The self-test for the integrity registers will leave the integrity registers in a known state.
- Testing the endorsement key pair integrity, if they exist. This requirement specifies that the TPM will verify that the endorsement key pair can sign and verify a known value. This test also tests the RSA sign and verify engine. If an endorsement key has not yet been generated the TPM action is manufacturer specific.
- The integrity of the protected capabilities of the TPM. This means that the TPM must ensure that its "microcode" has not changed, and not that a test must be run on each function.
- Any tamper-resistance markers. The tests on the tamper-resistance or tamper-evident markers are under programmable control. There is no requirement to check tamper-evident tape or the status of epoxy surrounding the case.

### 10.8.2 Recommended Checks

The TPM SHOULD check the following:

- The hash functionality. This check will hash a known value and compare it to an expected result. There is no requirement to accept external data to perform the check. The TPM MAY support a test using external data.
- Any symmetric algorithms. This check will use known data with a random key to encrypt and decrypt the data.
- Any additional asymmetric algorithms. This check will use known data to encrypt and decrypt.
- The key-wrapping mechanism. The TPM should wrap and unwrap a key. The TPM MUST NOT use the endorsement key pair for this test.

### 10.8.3 Self-Test Failure

When the TPM detects a failure during any self-test, the part experiencing the failure MUST enter a shut-down mode. This shut-down mode will allow only the following operation to occur:

- Update. The update function MAY replace invalid microcode, providing that the parts of the TPM that provide update functionality have passed self-test.

All other operations will return the error code TCPA_FAILEDSELFTEST.

## 10.9 Object Reuse

The TPM MUST destroy and erase all temporal objects when the TPM finishes processing the object. The use of an object can be a long-term operation. For instance, the TPM could load an identity key and keep the key in memory while performing multiple challenge and response operations. There is no requirement to unload the object after each operation, but there is a requirement that the object be properly disposed of when all operations are complete.

When an internal TPM process uses objects, no information regarding the object may be available to outside processes. The TPM MUST enforce access control to all objects carrying sensitive information.

## 10.10 Maintenance

*Start of informative comment:*

The maintenance feature is a vendor-specific feature, and its implementation is vendor-specific. The implementation must, however, meet the minimum requirements as defined in section 7.2.13 so that one implementation of the maintenance feature does not provide a hole into the TCPA system.

There is no requirement that the maintenance feature be available, but if it is implemented, then the requirements must be met.

The maintenance feature described in the specification is an example only, and not the only mechanism that a manufacturer could implement that meets these requirements.

*End of informative comment.*

The maintenance feature MUST ensure that the information can be on only one TPM at a time. Maintenance MUST ensure that at no time the process will expose a shielded location. Maintenance MUST require the active participation of the Owner.

## 10.11 Backup

*Start of informative comment:*

The purpose of backup is to take a key and move it to another TPM. The backup mechanism must move only migratable information.

The blob that the backup feature creates must be usable by any other TPM. This requirement holds only for keys and data that are usable by all TPMs. For example, there is no requirement that a 768-bit RSA key be acceptable by all TPM devices. The migration of information has a guarantee only when the key uses one of the required sizes.

*End of informative comment.*

The TPM MUST support the backup feature. The TPM MUST create a blob of migratable data that is readable by any other TPM. A receiving TPM MAY reject a backup blob if the underlying information is a non-standard size or algorithm.

## 10.12 Strength of Function

*Start of informative comment:*

The common criteria defines Strength of Function (SOF) as a qualification of a Target of Evaluation (TOE) security function expressing the minimum efforts assumed necessary to defeat its expected security behavior by directly attacking its underlying security mechanisms.

Here are some definitions for the common SOF criteria:

- **SOF-basic**: A level of the TOE SOF where analysis shows that the function provides adequate protection against casual breach of TOE security by attackers possessing a low attack potential.
- **SOF-medium**: A level of the TOE SOF where analysis shows that the function provides adequate protection against straightforward or intentional breach of TOE security by attackers possessing a moderate attack potential.
- **SOF-high**: A level of the TOE SOF where analysis shows that the function provides adequate protection against a deliberately planned or organized breach of TOE security by attackers possessing a high attack potential.

There is no single overall SOF definition; instead each operation needs a review of what the SOF should be. The Protection Profile will specify the SOF for each operation, command, function, and so on. For instance, the SOF for protection of the endorsement key pair will be SOF-high, but the SOF for tamper resistance will be SOF-basic.

The testing lab will determine if a specific security target implementation of the Protection Profile meets the SOF level. This specification will not specify definition of the SOF as this metric is an ever-changing value. That is, what was high a few years ago is now not even at the basic level. It is certainly possible that a device that receives certification will not pass given changes in the SOF definition in the future.

*End of informative comment.*

The TPM MUST report the SOF values to a Challenger and the SOF values MUST be part of the TPM endorsement certificate and the platform conformance certificate.

## 10.13 Physical Protection

*Start of informative comment:*

The main reason for inclusion of FIPS 140 is to specify the physical security requirements on a TPM. If a TPM manufacturer wishes to obtain full FIPS certification there are additional requirements that are not specified in the TCPA documentation.

*End of informative comment.*

TPM MUST satisfy the FIPS 140-1 (or it's successor) level 2 physical security requirements, or it's equivalent.

## 10.14 Protection Profile

*Start of informative comment:*

The TCPA specification will use two Protection Profiles to judge conformance with the specification. They are the TCPA Trusted Platform Module Protection Profile (TCPA-TPMPP) and the TCPA Trusted Platform Conection Protection Profile (TCPA-TPCPP).

The TPMPP provides the evaluation of a TPM; the security targets that reference this Protection Profile will provide the mechanism for platform manufacturers to judge between different TPM providers. The TOE for the TPMPP covers just the TPM and does not include any TSS functionality.

The TPCPP provides the evaluation of the connection of the TPM to the platform and the connection of the RMT to the platform and TPM. The security targets that reference this Protection Profile will provide the mechanism for platform purchasers the ability to judge between different platforms. The TOE for the TPCPP will include the TPMPP.

The Protection Profiles are separate documents and refer back to this specification. The following discussion of the Protection Profiles is for reference only and the actual text of the profiles supersedes any comments in this section.

The basis of the Protection Profiles is the attack tree that shows the threats against the TPM and TSS. The attack tree is a separate document that is an inherent part of this specification. The basic design

point for the attack tree is that the TPM should be resistant to all software attacks and somewhat resistant to hardware attacks.

*End of informative comment*

## 10.15 Compliance to Specification

*Start of informative comment:*

The TCPA does not evaluate compliance to this specification directly. The evaluation of compliance to the specification comes from the manufacturer creating a security target that meets the Protection Profile (either TPMPP or TPSPP).

After the TCPA creates a Protection Profile, each manufacturer has the option of creating a security target to evaluate against the Protection Profile. This security target is implementation specific and could cover either a machine or an application using the profile.

The evaluation of a security target provides assurances to the buying public that the manufacturer has created a secure interoperable system.

*End of informative comment*

## 10.16 Field Upgrade

*Start of informative comment:*

A TPM, once in the field, may have need to update the protected capabilities. This command, which is optional, provides the mechanism to perform the update.

*End of informative comment.*

The TPM SHOULD have provisions for upgrading the subsystem after shipment from the manufacturer. If provided the mechanism MUST follow the requirement from section 8.16 .

## 10.17 Physical Presence or Access

*Start of informative comment:*

This specification includes commands which require "local" or "physical" presence at the platform before the command will operate. The intention is that these commands cannot be activated without authorization provided by direct interaction with a person.

It must be possible to control a TPM. Such controls include those to clear an existing Owner from the TPM, temporarily deactivate a TPM and temporarily disable a TPM. Some such commands must work without conventional authorization information, because they will be required when the necessary authorization information is unavailable (because there is no Owner or because the authorization information has been lost). Such commands are subject to "denial of service" attacks, and ideally require other forms of authorization.

Some commands are therefore prescribed to require physical presence (of a person) at the platform before the command will operate. Such commands could be authorised with or by purely physical or electrical methods, or with or by physical presence detected using software when the platform is in a restricted state. Such authorization is difficult or impossible to reproduce by rogue software, depending on the exact method of implementation. The actual method of implementation of such authorization is the choice of the manufacturer. The overall strength of such authorization is reflected in the "security target" of the platform.

In a PC, such authorization might be implemented using direct electrical connections from a switch, or using software during the POST.

*End of informative comment.*

The requirement for physical presence MUST be met by the platform manufacturer using some physical mechanism.

### 10.17.1 TSC_PhysicalPresence

*Start of informative comment:*

Some TPM operations require an indication of an owner's physical presence at the platform. These are administrative operations that need to function when the owner's authentication materials are not available. An indication of physical presence is an alternate method for proving ownership of the platform. Generally this is implemented using a hardware signal generated as a result of an owner's physical action such as changing an internal switch, jumper, or button. However, the architecture or design of some platforms prevent this from being a cost effective implementation.

This operation provides a method for the platform to provide proof of physical presence using the state of the platform and user action. The platform has the option to attach a hardware signaling mechanism to the TPM or use this command in the absence or in conjunction with a hardware signal.

The values of the PhysicalPresence and PhysicalPresenceLock flags are preserved by TPM_SaveState and TPM_Startup(stType = TCPA_ST_STATE) to prevent changing the flag while in any of the platform's power suspend states.

Note: This operation does not affect the state of the indication of unambiguous physical presence which may be the same or same hardware signal, depending on implementation.

While not a requirement, it is likely the following flags will be set by the Platform manufacturer in a single operation prior to shipment to the owner:

- physicalPresenceLifetimeLock = TRUE
- physicalPresenceHWEnable = Design and owner requirements dependent, and
- physicalPresenceCMDEnable = Design and owner requirements dependent.

*End of informative comment.*

**Type**

TCPA connection capability. Optional function this functionality can be implemented by any vendor specific command

**Incoming Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TCPA_COMMAND_CODE | ordinal | Command ordinal, fixed value of TSC_ORD_PhysicalPresence. |
| 4 | 2 | | | TCPA_PHYSICAL_PRESENCE | physicalPresence | The state to set the TPM's Physical Presence flags. |

**Outgoing Operands and Sizes**

| PARAM # | PARAM SZ | HMAC # | HMAC SZ | Type | Name | Description |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TCPA_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TCPA_RESULT | returnCode | The return code of the operation. See section 4.3 of Main Specification. |

**Descriptions**

This command must implemented in the TPM, however support for all of the bits is optional.

The operation sets the state of the physicalPresenceLifetimeLock, physicalPresenceHWEnable, and physicalPresenceCMDEnable flags to indicate how physical presence is to be indicated. It also sets the PhysicalPresence and PhysicalPresenceLock flags, if enabled, during operation of the Platform to indicate physical presence. This is a bit mask allowing a combination of flags to be set in a single operation.

Note: The TPM_PhysicalEnable requires unambiguous evidence of the presence of physical access. This is a higher level of proof than the other "physical presence" commands. A PhysicalPresence flag set to TRUE, SHALL NOT be sufficient proof to permit execution of TPM_PhysicalEnable unless it is impossible for software to subvert the TSC_PhysicalPresence command.

**Actions**

1. This operation MUST be implemented to process the values in the following order:
   a. physicalPresenceHWEnable and physicalPresenceCMDEnable
   b. physicalPresenceLifetimeLock
   c. PhysicalPresence
   d. PhysicalPresenceLock
2. Once the PhysicalPresenceLock flag is set to TRUE, the TPM MUST not modify the PhysicalPresence flag until a TPM_Init followed by TPM_Startup(stType = TCPA_ST_CLEAR). Upon a TPM_Init and TPM_Startup(stType = TCPA_ST_STATE) the TPM MUST set the PhysicalPresenceLock flag to FALSE.
3. If the PhysicalPresenceLock flag is set to TRUE upon any call to this operation, the TPM MUST cause no action and MUST return the error TCPA_BAD_PARAMETER.

## 10.18 Other Specifications

*Start of informative comment:*

There are other security specifications and this section describes them and what level of compliance the TCPA may have with them.

- Rainbow Series: The Rainbow Series of specifications is being phased out by Protection Profiles. There is no requirement that the TCPA be Orange Book compatible.
- ITSEC: ITSEC is a European standard that is being phased out by Protection Profiles. There is no requirement that TCPA use any ITSEC specifications.
- FIPS: The FIPS 140 specification covers cryptographic modules and the hardware implementation of these modules. In many ways, Protection Profiles and FIPS overlap. Some of the FIPS 140 requirements are specified in this specification; however, compliance with the entire specification is not required.

*End of informative comment.*

Individual manufacturers MAY do the additional design and testing to obtain a FIPS 140 certification, but there is no requirement that a TCPA device obtain this testing.

Specifications or standards included in this specification

- PKCS#1: RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS) Version 2.0
    - RSAES_OAEP (2.0)
    - RSASSA-PKCS1-v1_5
- ITU-T Recommendation X.509 | ISO/IEC 9594-8: "Information technology - Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks", 4th Edition.
- DES/3DES: Data Encryption Standard FIPS 46-3 (DES) : National Institute of Standards and Technology
- ASN.1: Abstract Syntax Notation One : ITU-T Recommendations X.680-X.683
- FIPS 140-1: Federal Information Processing Standards Publication 140-1 "Security Requirements for Cryptographic Modules"
- BER: Basic Encoding Rules : ITU-T Recommendation X.690-691 (1997)
- ISO 15408 (Common Criteria)
- SHA-1: Secure Hash Algorithm : NIST FIPS PUB 180-1, "Secure Hash Standard," : National Institute of Standards and Technology
- RFC 2104 (HMAC)

# Appendix A: Glossary

**3DES**

DES using a key of a size that is 3X the size that of a DES key. See **DES**.

**Blob**

Opaque data of fixed or variable size. The meaning and interpretation of the data is outside the scope and context of the Subsystem.

**Challenger**

An entity that requests and has the ability to interpret integrity metrics from a Subsystem.

**Conformance Credential**

A credential that states the conformance to the TCPA specification of: the TPM; the method of incorporation of the TPM into the platform; the RTM; and the method of incorporation of the RTM into the platform.

**Denial-of-service attack**

A attack on a system (or subsystem) which has no affect on information except to prevent its use.

**DES**

Symmetric key encryption using a key size of 56 bits defined by NIST as FIPS 46-3. Reference http://csrc.ncsl.nist.gov/cryptval/des.htm.

**Endorsement Credential**

A credential containing a public key (the endorsement public key) that was generated by a genuine TPM.

**Endorsement Key**

A term used ambiguously, depending on context, to mean a pair of keys, or the public key of that pair, or the private key of that pair; an asymmetric key pair generated by a TPM that is used as proof that a TPM is a genuine TPM; the public endorsement key (PUBEK); the private endorsement key (PRIVEK).

**Identity Credential**

A credential issued by a Privacy CA that provides an identity for the TPM.

**Integrity metric(s)**

Values that are the results of measurements on the integrity of the platform.

**Man-in-the-middle attack**

An attack by an entity intercepting communications between two others without their knowledge and by intercepting that communication is able to obtain or modify the information between them.

**Migratable**

A key which may be transported outside the specific TPM.

**Non-Migratable**

A key which cannot be transported outside a specific TPM; a key that is (statistically) unique to a particular TPM.

**Non-Volatile**

Storage location or memory that retain their values after power-off or a TPM_Init function.

**Owner**

The entity that owns the platform in which a TPM is installed. Since there is, by definition, a one-to-one relationship between the TPM and the platform, the Owner is also the Owner of the TPM. The Owner of

the platform is not necessarily the "user" of the platform (e.g., in a corporation, the Owner of the platform might be the IT department while the user is an employee.) The Owner has administration rights over the TPM.

**PKI Identity Protocol**

The protocol used to insert anonymous identities into the TPM.

**Platform Credential**

A credential that states that a specific platform contains a genuine TCPA Subsystem.

**POST**

POST refers to the Power On Self Test performed by a PC.

**Protection Profile**

A document that defines all attacks and how they are resisted by the TPM, the RTM, and the methods by which they are incorporated into the platform.

**Privacy CA**

An entity that issues an Identity Credential for a TPM based on trust in the entities that vouch for the TPM via the Endorsement Credential, the Conformance Credential, and the Platform Credential.

**Private Endorsement Key** (PRIVEK)

The private key of the key pair that proves that a TPM is a genuine TPM. The PRIVEK is (statistically) unique to only one TPM.

**Public Endorsement Key** (PUBEK)

A public key that proves that a TPM is a genuine TPM. The PUBEK is (statistically) unique to only one TPM.

**Random number generator (RNG)**

A pseudo-random number generator that must be initialized with unpredictable data and provides, "random" numbers on demand.

**Root of Trust for Measurement (RTM)**

The point from which all trust in the measurement process is predicated. The RTM contains many components to provide this level of trust. The design document shows that the RTM includes a core component, the computing engine to run the core component, physical connections of the core and the computing engine and other items.

**Root of Trust for Reporting (RTR)**

The point from which all trust in reporting of measured information is predicated.

**Root of Trust for Storing (RTS)**

The point from which all trust in Protected Storage is predicated.

**RSA**

An (asymmetric) encryption method using two keys: a private key and a public key. Reference: http://www.rsa.com .

**SHA-1**

A NIST defined hashing algorithm producing a 160 bit result from an arbitrary sized source as specified in FIPS 180-1. Reference: http://csrc.ncsl.nist.gov/cryptval/shs.html.

**Storage Root Key (SRK)**

The root key of a hierarchy of keys associated with a TPM; generated within a TPM; a non-migratable key.

**Subsystem**

The combination of the TSS and the TPM.

**Support Services (TSS)**

Services to support the TPM but which do not need the protection of the TPM. The same as **Trusted Platform Support Services.**

**Trusted Building Block (TBB)**

A trusted Platform is instantiated as a Trusted Building Block (TBB) which is the evaluated component of a trusted system. The TBB is composed of the TPM, the Core RTM and the connection between them.

**TCPA-protected capability**

A function which is protected within the TPM, and has access to TPM secrets.

**TPM Identity**

One of the anonymous PKI identities belonging to a TPM; a TPM may have multiple identities.

**TPM POST**

TPM POST refers to the Power On Self Test performed by a TPM.

**Trusted Platform Agent (TPA)**

Trusted Platform Agent; the component within the platform that reports integrity metrics, logs, Validation Data, etc. to a Challenger; outside the scope of this specification.

**Trusted Platform Measurement Store (TPMS)**

Storage locations within the Subsystem, which contain unprotected logs of measurement process.

**Trusted Platform Module (TPM)**

The set of functions and data that are common to all types of platform, which must be trustworthy if the Subsystem is to be trustworthy; a logical definition in terms of protected capabilities and shielded locations.

**Trusted Platform Support Services (TSS)**

The set of functions and data that are common to all types of platform, which are not required to be trustworthy (and therefore do not need to be part of the TPM).

**User**

An entity that uses the platform in which a TPM is installed. The only rights that a User has over a TPM are the rights given to the User by the Owner. These rights are expressed in the form of authorization data, given by the Owner to the User, that permits access to entities protected by the TPM. The User of the platform is not necessarily the "owner" of the platform (e.g., in a corporation, the owner of the platform might be the IT department while the User is an employee). There can be multiple Users.

**Validation Credential**

A credential that states values of measurements that should be obtained when measuring a particular part of the platform when the part is functioning as expected.

**Validation Data**

Data inside a Validation Credential; the values that the integrity measurements should produce when the part of a platform described by the Validation Credential is working correctly.

**Validation Entity**

An entity that issues a Validation Certificate for a component; the manufacturer of that component; an agent of the manufacturer of that component.

**Volatile**

Storage locations or memory that are either set to a predefined value (e.g.,zero) or have values that are undefined upon completion of a power-on or TPM_Init function.

# Appendix B: Key Usage Table

This table summarizes the types of keys associated with a given TPM command.

| | | | | | First Key | | | | | | Second Key | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Section | Name | First Key | Second Key | SIGNING | STORAGE | IDENTITY | AUTHCHG | BIND | LEGACY | SIGNING | STORAGE | IDENTITY | AUTHCHG | BIND | LEGACY |
| 5.6.1 | TPM_ChangeAuth | parent | blob | | | x | | | | | x | x | x | | x | x |
| 5.2.5 | TPM_OSAP | entity | | | x | x | x | x | x | x | | | | | | |
| 5.7.1 | TPM_ChangeAuthAsymStart | idKey | ephemeral | | | | x | | | | | | | x | | |
| 5.7.2 | TPM_ChangeAuthAsymFinish | parent | ephemeral | | | x | | | | | | | | x | | |
| 6.3.3 | TPM_Quote | key | | | x | | x | | | x | | | | | | |
| 7.2.1 | TPM_Seal | key | | | | x | | | | | | | | | | |
| 7.2.2 | TPM_Unseal | parent | | | | x | | | | | | | | | | |
| 7.2.4 | TPM_UnBind | key | | | | | | | x | x | | | | | | |
| 7.2.5 | TPM_CreateWrapKey | parent | | | | x | | | | | | | | | | |
| 7.2.8 | TPM_LoadKey | parent | inKey | | | x | | | | | x | x | x | | x | x |
| 7.2.10 | TPM_GetPubKey | key | | | x | x | x | x | x | x | | | | | | |
| 7.2.11 | TPM_CreateMigrationBlob | parent | blob | | | x | | | | | x | x | | | x | x |
| 7.2.12 | TPM_ConvertMigrationBlob | parent | | | | x | | | | | | | | | | |
| 8.3.1 | TPM_CertifyKey | certKey | inKey | | x | | x | | | x | x | x | x | | x | x |
| 8.7.1 | TPM_Sign | key | | | x | | | | | x | | | | | | |
| 8.9.2 | TPM_CertifySelfTest | key | | | x | | x | | | x | | | | | | |
| 8.11.2 | TPM_GetCapabilitySigned | key | | | x | | x | | | x | | | | | | |
| 8.12.2 | TPM_GetAuditEventSigned | key | | | x | | x | | | x | | | | | | |
| 9.3.4 | TPM_ActivateIdentity | idKey | | | | | x | | | | | | | | | |